CreateAssisgnment.tsx

```
import { useState } from "react";
import { useNavigate } from "react-router-dom";
import { Button } from "@/components/ui/button";
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from "@/components/ui/card";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import { Textarea } from "@/components/ui/textarea";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@/components/ui/select";
import { Navigation } from "@/components/Navigation";
import { FileUpload, FileAttachment } from "@/components/FileUpload";
import { useAuth } from "@/hooks/useAuth";
import { toast } from "@/hooks/use-toast";
import { ArrowLeft, Save, Calendar, DollarSign } from "lucide-react";

const subjects = [
  "Computer Science",
  "Mathematics",
  "Physics",
  "Chemistry",
  "Biology",
  "Engineering",
  "Business",
  "Economics",
  "Psychology",
  "Literature",
  "History",
  "Political Science",
  "Sociology",
```

```
  "Philosophy",
  "Art & Design",
  "Other"
];

export default function CreateAssignment() {
  const { user } = useAuth();
  const navigate = useNavigate();
  const [isSubmitting, setIsSubmitting] = useState(false);
  const [files, setFiles] = useState<File[]>([]);
  const [formData, setFormData] = useState({
    title: "",
    description: "",
    requirements: "",
    subject: "",
    price: "",
    deadline: "",
  });

  const handleInputChange = (field: string, value: string) => {
    setFormData(prev => ({ ...prev, [field]: value }));
  };

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!user) return;

    // Validation
    if (!formData.title.trim()) {
```

```
  toast({
    title: "Missing title",
    description: "Please provide a title for your assignment",
    variant: "destructive",
  });
  return;
}

if (!formData.description.trim()) {
  toast({
    title: "Missing description",
    description: "Please provide a description for your assignment",
    variant: "destructive",
  });
  return;
}

if (!formData.subject) {
  toast({
    title: "Missing subject",
    description: "Please select a subject",
    variant: "destructive",
  });
  return;
}

const price = parseFloat(formData.price);
if (isNaN(price) || price <= 0) {
  toast({
```

```
    title: "Invalid price",
    description: "Please enter a valid price",
    variant: "destructive",
  });
  return;
}

const deadline = new Date(formData.deadline);
if (deadline <= new Date()) {
  toast({
    title: "Invalid deadline",
    description: "Deadline must be in the future",
    variant: "destructive",
  });
  return;
}

setIsSubmitting(true);

try {
  const formDataToSend = new FormData();
  formDataToSend.append("title", formData.title.trim());
  formDataToSend.append("description", formData.description.trim());
  formDataToSend.append("requirements", formData.requirements.trim());
  formDataToSend.append("subject", formData.subject);
  formDataToSend.append("price", price.toString());
  formDataToSend.append("deadline", deadline.toISOString());
  formDataToSend.append("studentId", user.id);
  formDataToSend.append("studentName", user.name);
```

```
  files.forEach((file) => {
    formDataToSend.append("attachments", file);
  });

  const response = await fetch("http://localhost:5000/api/assignments", {
    method: "POST",
    headers: {
      Authorization: `Bearer ${localStorage.getItem("token")}`,
    },
    body: formDataToSend,
  });

  if (!response.ok) {
    throw new Error("Failed to post assignment");
  }

  const data = await response.json();
  toast({
    title: "Assignment posted!",
    description: "Your assignment has been submitted for review",
  });
  navigate("/assignments");
 } catch (error) {
  toast({
    title: "Failed to post assignment",
    description: error instanceof Error ? error.message : "Please try again",
    variant: "destructive",
  });
```

```
  } finally {
    setIsSubmitting(false);
  }
};

// Set minimum date to tomorrow
const minDate = new Date();
minDate.setDate(minDate.getDate() + 1);
const minDateString = minDate.toISOString().split("T")[0];

return (
  <div className="min-h-screen bg-background">
    <Navigation />
    <div className="container mx-auto px-4 py-8 max-w-3xl">
      {/* Header */}
      <div className="flex flex-col gap-4 mb-6">
        <Button
          variant="ghost"
          onClick={() => navigate(-1)}
          className="flex items-center self-start"
        >
          <ArrowLeft className="h-4 w-4 mr-2" />
          Back
        </Button>
        <div>
          <h1 className="text-2xl sm:text-3xl font-bold">Post New Assignment</h1>
          <p className="text-muted-foreground text-sm sm:text-base">
            Provide details about your assignment to get expert help
          </p>
```

```
  </div>
</div>
<form onSubmit={handleSubmit}>
  <Card className="shadow-elegant">
    <CardHeader>
      <CardTitle>Assignment Details</CardTitle>
      <CardDescription>
        Fill in all the required information to help administrators understand your needs
      </CardDescription>
    </CardHeader>
    <CardContent className="space-y-6">
      {/* Title */}
      <div className="space-y-2">
        <Label htmlFor="title">Assignment Title *</Label>
        <Input
          id="title"
          value={formData.title}
          onChange={(e) => handleInputChange("title", e.target.value)}
          placeholder="e.g., React Application Development"
          required
        />
      </div>
      {/* Subject and Price Row */}
      <div className="grid gap-4 md:grid-cols-2">
        <div className="space-y-2">
          <Label htmlFor="subject">Subject *</Label>
          <Select
            value={formData.subject}
            onValueChange={(value) => handleInputChange("subject", value)}
```

```
              required
            >
              <SelectTrigger>
                <SelectValue placeholder="Select subject" />
              </SelectTrigger>
              <SelectContent>
                {subjects.map((subject) => (
                  <SelectItem key={subject} value={subject}>
                    {subject}
                  </SelectItem>
                ))}
              </SelectContent>
            </Select>
          </div>
          <div className="space-y-2">
            <Label htmlFor="price">Budget (USD) *</Label>
            <div className="relative">
              <DollarSign className="absolute left-3 top-1/2 transform -translate-y-1/2 h-4 w-4 text-muted-foreground" />
              <Input
                id="price"
                type="number"
                step="0.01"
                min="1"
                value={formData.price}
                onChange={(e) => handleInputChange("price", e.target.value)}
                placeholder="100.00"
                className="pl-9"
                required
```

```
            />
          </div>
        </div>
      </div>
      {/* Deadline */}
      <div className="space-y-2">
        <Label htmlFor="deadline">Deadline *</Label>
        <div className="relative">
          <Calendar className="absolute left-3 top-1/2 transform -translate-y-1/2 h-4 w-4 text-muted-foreground" />
          <Input
            id="deadline"
            type="date"
            min={minDateString}
            value={formData.deadline}
            onChange={(e) => handleInputChange("deadline", e.target.value)}
            className="pl-9"
            required
          />
        </div>
      </div>
      {/* Description */}
      <div className="space-y-2">
        <Label htmlFor="description">Description *</Label>
        <Textarea
          id="description"
          value={formData.description}
          onChange={(e) => handleInputChange("description", e.target.value)}
          placeholder="Provide a detailed description of your assignment..."
```

```
    rows={4}
    required
  />
</div>
{/* Requirements */}
<div className="space-y-2">
  <Label htmlFor="requirements">Specific Requirements</Label>
  <Textarea
    id="requirements"
    value={formData.requirements}
    onChange={(e) => handleInputChange("requirements", e.target.value)}
    placeholder="List any specific requirements, technologies, formatting guidelines, etc."
    rows={3}
  />
</div>
{/* File Upload */}
<div className="space-y-2">
  <Label>Supporting Materials</Label>
  <FileUpload
    files={files}
    onFilesChange={setFiles}
    maxFiles={10}
    maxSize={10 * 1024 * 1024} // 10MB
  />
  <p className="text-sm text-muted-foreground">
    Upload any relevant files, documents, or reference materials
  </p>
</div>
{/* Submit Button */}
```

```
      <div className="flex justify-end space-x-4 pt-6 border-t">
       <Button
        type="button"
        variant="outline"
        onClick={() => navigate(-1)}
       >
        Cancel
       </Button>
       <Button
        type="submit"
        disabled={isSubmitting}
        className="bg-gradient-primary hover:shadow-glow"
       >
        {isSubmitting ? (
         "Posting..."
        ) : (
         <>
          <Save className="h-4 w-4 mr-2" />
          Post Assignment
         </>
        )}
       </Button>
      </div>
     </CardContent>
    </Card>
   </form>
   {/* Help Card */}
   <Card className="mt-6">
    <CardContent className="pt-6">
```

```
          <h3 className="font-semibold mb-2">Tips for better results:</h3>
          <ul className="text-sm text-muted-foreground space-y-1">
            <li>• Be specific about your requirements and expectations</li>
            <li>• Include all relevant materials and reference documents</li>
            <li>• Set a realistic deadline that allows for quality work</li>
            <li>• Provide clear grading criteria or rubrics if available</li>
            <li>• Mention your academic level and course context</li>
          </ul>
        </CardContent>
      </Card>
    </div>
  </div>
 );
}
```

Backend index.js

```
const express = require("express");
const mysql = require("mysql2/promise");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const bodyParser = require("body-parser");
const cors = require("cors");
const multer = require("multer");
const path = require("path");
const fs = require("fs");
const nodemailer = require("nodemailer");

const app = express();
```

```
const port = 5000;

// Middleware
app.use(cors());
app.use(bodyParser.json());

// Configure multer for file uploads
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    const uploadDir = path.join(__dirname, "uploads");
    if (!fs.existsSync(uploadDir)) {
      fs.mkdirSync(uploadDir);
    }
    cb(null, uploadDir);
  },
  filename: (req, file, cb) => {
    cb(null, `${Date.now()}-${file.originalname}`);
  },
});

const upload = multer({ storage });

// Database connection
const dbConfig = {
  host: "localhost",
  user: "root",
  password: "",
  database: "assignment_help_hub",
};
```

```
let connection;
mysql
  .createConnection(dbConfig)
  .then((conn) => {
    connection = conn;
    console.log("Connected to MySQL database");
  })
  .catch((err) => {
    console.error("Error connecting to MySQL database", err);
  });

// Secret key for JWT
const JWT_SECRET =
"3q2h8i7s6e5r4t3y8u1i9o8p7o6i5u4y3t2r1e0w9q8e7r6t9y4u3i2o1p0i9u8y7t6r5e4w3q2e1r";

// Function to generate a random password
function generateRandomPassword() {
  const length = 10;
  const charset = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
  let password = "";
  for (let i = 0, n = charset.length; i < length; ++i) {
    password += charset.charAt(Math.floor(Math.random() * n));
  }
  return password;
}

// Configure nodemailer to send emails
const transporter = nodemailer.createTransport({
```

```
  service: "gmail",
  auth: {
    user: "assignmenthelphub.service@gmail.com",
    pass: "hhbs zgsb mhhc mxrr",
  },
});

// Middleware to authenticate JWT
const authenticateJWT = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1]; // Extract the token from the Authorization header

  if (!token) {
    return res.status(401).json({ error: "Unauthorized" });
  }

  try {
    const decoded = jwt.verify(token, JWT_SECRET);
    req.user = decoded;
    next();
  } catch (error) {
    console.error("Error authenticating token:", error);
    res.status(401).json({ error: "Unauthorized" });
  }
};

// Routes
app.post("/api/register", async (req, res) => {
```

```
const { username, email, password, role, name } = req.body;
try {
  const hashedPassword = await bcrypt.hash(password, 10);
  const [result] = await connection.query(
    "INSERT INTO users (username, email, password, role, name) VALUES (?, ?, ?, ?, ?)",
    [username, email, hashedPassword, role, name]
  );

  const [users] = await connection.query("SELECT * FROM users WHERE id = ?", [result.insertId]);
  if (users.length === 0) {
    return res.status(500).json({ error: "User registration failed" });
  }

  const user = users[0];
  const token = jwt.sign(
    {
      id: user.id,
      username: user.username,
      email: user.email,
      role: user.role,
      name: user.name,
      createdAt: user.createdAt,
    },
    JWT_SECRET,
    { expiresIn: "1h" }
  );

  res.status(201).json({ token });
} catch (error) {
```

```
    console.error("Error registering user:", error);
    res.status(500).json({ error: "Error registering user" });
  }
});

app.post("/api/login", async (req, res) => {
  const { username, password } = req.body;
  try {
    const [users] = await connection.query("SELECT * FROM users WHERE username = ?", [username]);
    if (users.length === 0) {
      return res.status(401).json({ error: "Invalid credentials" });
    }

    const user = users[0];
    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) {
      return res.status(401).json({ error: "Invalid credentials" });
    }

    const token = jwt.sign(
      {
        id: user.id,
        username: user.username,
        email: user.email,
        role: user.role,
        name: user.name,
        createdAt: user.createdAt,
      },
      JWT_SECRET,
```

```
      { expiresIn: "1h" }
    );

    res.status(200).json({ token });
  } catch (error) {
    console.error("Error logging in:", error);
    res.status(500).json({ error: "Error logging in" });
  }
});

app.post("/api/forgot-password", async (req, res) => {
  const { email } = req.body;
  try {
    const [users] = await connection.query("SELECT * FROM users WHERE email = ?", [email]);
    if (users.length === 0) {
      return res.status(404).json({ error: "Email not found" });
    }

    const user = users[0];
    const newPassword = generateRandomPassword();
    const hashedPassword = await bcrypt.hash(newPassword, 10);

    await connection.query("UPDATE users SET password = ? WHERE email = ?", [hashedPassword,
email]);

    const currentYear = new Date().getFullYear();
    const mailOptions = {
      from: "assignmenthelphub.service@gmail.com",
      to: email,
```

```
    subject: "Your New Password",
    html: `
      <!DOCTYPE html>
      <html lang="en">
      <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Your New Password</title>
        <style>
          body { font-family: Arial, sans-serif; background-color: #f4f4f4; color: #333; line-height: 1.6; margin: 0; padding: 0; }
          .container { max-width: 600px; margin: 20px auto; padding: 20px; background-color: #fff; border-radius: 5px; box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); border: 1px solid #B0C4DE; }
          .header { text-align: center; padding-bottom: 10px; border-bottom: 1px solid #B0C4DE; }
          .header h1 { color: #B0C4DE; }
          .content { padding: 20px 0; }
          .content p { margin-bottom: 15px; }
          .password { background-color: #B0C4DE; color: #fff; padding: 10px; text-align: center; font-size: 18px; border-radius: 5px; margin: 20px 0; }
          .footer { text-align: center; padding-top: 10px; border-top: 1px solid #B0C4DE; color: #B0C4DE; }
        </style>
      </head>
      <body>
        <div class="container">
          <div class="header">
            <h1>Password Reset</h1>
          </div>
          <div class="content">
            <p>Hello,</p>
```

```
            <p>We have generated a new password for you as requested. Please use the following password to log in to your account:</p>
            <div class="password">
              ${newPassword}
            </div>
            <p>For security purposes, we recommend that you update your password immediately after logging in. You can do this by navigating to your profile settings.</p>
            <p>If you did not request a password reset, please contact our support team immediately.</p>
            <p>Support: assignmenthelphub.service@gmail.com</p>
            <p>Thank you!</p>
          </div>
          <div class="footer">
            <p>&copy; ${currentYear} Assignment Help Hub. All rights reserved.</p>
          </div>
        </div>
      </body>
      </html>
    `,
  };

  await transporter.sendMail(mailOptions);
  res.status(200).json({ message: "New password sent to your email" });
 } catch (error) {
  console.error("Error resetting password:", error);
  res.status(500).json({ error: "Error resetting password" });
 }
});

// Assignment routes
```

```
app.get("/api/assignments", authenticateJWT, async (req, res) => {
  try {
    const [assignments] = await connection.query("SELECT * FROM assignments");
    res.status(200).json(assignments);
  } catch (error) {
    console.error("Error fetching assignments:", error);
    res.status(500).json({ error: "Error fetching assignments" });
  }
});

app.get("/api/assignments/:id", authenticateJWT, async (req, res) => {
  const { id } = req.params;
  try {
    const [assignments] = await connection.query("SELECT * FROM assignments WHERE id = ?", [id]);
    if (assignments.length === 0) {
      return res.status(404).json({ error: "Assignment not found" });
    }
    res.status(200).json(assignments[0]);
  } catch (error) {
    console.error("Error fetching assignment:", error);
    res.status(500).json({ error: "Error fetching assignment" });
  }
});

app.post("/api/assignments", authenticateJWT, upload.array("attachments"), async (req, res) => {
  const { title, description, requirements, price, deadline, subject, studentId, studentName } = req.body;
  try {
    const [result] = await connection.query(
```

```
    "INSERT INTO assignments (title, description, requirements, price, originalPrice, deadline, subject, studentId, studentName, status) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, 'pending')",
    [title, description, requirements, price, price, deadline, subject, studentId, studentName]
   );

   const assignmentId = result.insertId;
   if (req.files && req.files.length > 0) {
    for (const file of req.files) {
     await connection.query(
      "INSERT INTO files (name, type, size, url, assignmentId) VALUES (?, ?, ?, ?, ?)",
      [file.originalname, file.mimetype, file.size, file.path, assignmentId]
     );
    }
   }

   const [assignments] = await connection.query("SELECT * FROM assignments WHERE id = ?", [assignmentId]);
   res.status(201).json(assignments[0]);
  } catch (error) {
   console.error("Error creating assignment:", error);
   res.status(500).json({ error: "Error creating assignment" });
  }
});

app.put("/api/assignments/:id", authenticateJWT, upload.array("attachments"), async (req, res) => {
  const { id } = req.params;
  const { title, description, requirements, price, deadline, subject, status, adminId, adminName, rejectionReason, isPaid } = req.body;
  try {
   await connection.query(
```

```
    "UPDATE assignments SET title = ?, description = ?, requirements = ?, price = ?, deadline = ?, subject = ?, status = ?, adminId = ?, rejectionReason = ?, isPaid = ? WHERE id = ?",
    [title, description, requirements, price, deadline, subject, status, adminId, rejectionReason, isPaid, id]
   );

   if (req.files && req.files.length > 0) {
    for (const file of req.files) {
     await connection.query(
      "INSERT INTO files (name, type, size, url, assignmentId) VALUES (?, ?, ?, ?, ?)",
      [file.originalname, file.mimetype, file.size, file.path, id]
     );
    }
   }

   const [assignments] = await connection.query("SELECT * FROM assignments WHERE id = ?", [id]);
   res.status(200).json(assignments[0]);
  } catch (error) {
   console.error("Error updating assignment:", error);
   res.status(500).json({ error: "Error updating assignment" });
  }
});

app.delete("/api/assignments/:id", authenticateJWT, async (req, res) => {
  const { id } = req.params;
  try {
   await connection.query("DELETE FROM assignments WHERE id = ?", [id]);
   res.status(200).json({ message: "Assignment deleted successfully" });
  } catch (error) {
   console.error("Error deleting assignment:", error);
```

```
    res.status(500).json({ error: "Error deleting assignment" });
  }
});

// File routes
app.get("/api/files/:id", authenticateJWT, async (req, res) => {
  const { id } = req.params;
  try {
    const [files] = await connection.query("SELECT * FROM files WHERE id = ?", [id]);
    if (files.length === 0) {
      return res.status(404).json({ error: "File not found" });
    }
    res.status(200).json(files[0]);
  } catch (error) {
    console.error("Error fetching file:", error);
    res.status(500).json({ error: "Error fetching file" });
  }
});

// Submission routes
app.post("/api/submissions", authenticateJWT, upload.array("files"), async (req, res) => {
  const { assignmentId, adminId, adminName, description } = req.body;
  try {
    const [result] = await connection.query(
      "INSERT INTO submissions (assignmentId, adminId, adminName, description) VALUES (?, ?, ?, ?)",
      [assignmentId, adminId, adminName, description]
    );

    const submissionId = result.insertId;
```

```
    if (req.files && req.files.length > 0) {
      for (const file of req.files) {
        await connection.query(
          "INSERT INTO files (name, type, size, url, submissionId) VALUES (?, ?, ?, ?, ?)",
          [file.originalname, file.mimetype, file.size, file.path, submissionId]
        );
      }
    }

    const [submissions] = await connection.query("SELECT * FROM submissions WHERE id = ?", [submissionId]);
    res.status(201).json(submissions[0]);
  } catch (error) {
    console.error("Error creating submission:", error);
    res.status(500).json({ error: "Error creating submission" });
  }
});

// Negotiation routes
app.post("/api/negotiations", authenticateJWT, async (req, res) => {
  const { assignmentId, proposedPrice, message } = req.body;
  const fromUserId = req.user.id;
  const fromUserName = req.user.name;
  const fromUserRole = req.user.role;
  try {
    const [result] = await connection.query(
      "INSERT INTO negotiations (assignmentId, fromUserId, fromUserName, fromUserRole, proposedPrice, message) VALUES (?, ?, ?, ?, ?, ?)",
      [assignmentId, fromUserId, fromUserName, fromUserRole, proposedPrice, message]
```

```
    );

    const [negotiations] = await connection.query("SELECT * FROM negotiations WHERE id = ?", [result.insertId]);
    res.status(201).json(negotiations[0]);
  } catch (error) {
    console.error("Error creating negotiation:", error);
    res.status(500).json({ error: "Error creating negotiation" });
  }
});

// Payment method routes
app.get("/api/payment-methods", authenticateJWT, async (req, res) => {
  try {
    const [paymentMethods] = await connection.query("SELECT * FROM payment_methods WHERE isAvailable = TRUE");
    res.status(200).json(paymentMethods);
  } catch (error) {
    console.error("Error fetching payment methods:", error);
    res.status(500).json({ error: "Error fetching payment methods" });
  }
});

app.post("/api/payment-methods", authenticateJWT, async (req, res) => {
  const { name, type, details } = req.body;
  try {
    const [result] = await connection.query(
      "INSERT INTO payment_methods (name, type, details) VALUES (?, ?, ?)",
      [name, type, details]
```

```
    );

    const [paymentMethods] = await connection.query("SELECT * FROM payment_methods WHERE id =
?", [result.insertId]);
    res.status(201).json(paymentMethods[0]);
  } catch (error) {
    console.error("Error creating payment method:", error);
    res.status(500).json({ error: "Error creating payment method" });
  }
});

// Start server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

So why is the token retrieved from localstorage null, this makes file upload fail with jwt “token malformed error”