**ProjectPreviewModal.jsx:**

import React, { useState } from 'react';

import './ProjectPreviewModal.css';

import ProjectAssessmentModal from '../popups/ProjectAssessmentModal';

import { toast } from 'react-toastify';

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

function ProjectPreviewModal({ show, onClose, project, onDelete }) {

  const [showAssessmentModal, setShowAssessmentModal] = useState(false);

  if (!show || !project) return null;

  const handleDelete = () => {

    // Display a confirmation toast

    const toastId = toast.loading('Are you sure you want to delete this project?', {

      position: 'top-center',

      autoClose: 5000, // Keeps the toast open

      closeButton: false,

      draggable: false,

      theme: 'light',

    });

    // Show confirmation buttons inside the toast

    toast.update(toastId, {

      render: (

        <div>

          <p>Click to confirm or cancel deletion.</p>

          <div style={{ display: 'flex', justifyContent: 'space-around' }}>

            <button

              onClick={() => {

                onDelete(project.\_id); // Delete the project

                toast.success('Project deleted successfully!', {

                  position: 'top-center',

                   autoClose: 3000,  // Toast will disappear in 1 seconds

                  closeButton: false,

                  draggable: false,

                  theme: 'light',

                });

                toast.dismiss(toastId); // Close the toast after confirming

                onClose(); // Close the modal after confirmation

              }}

              style={{ padding: '5px 10px', backgroundColor: '#9D0E0F', color: 'white' , border: 'none',borderRadius: '5px'}}

            >

              Confirm

            </button>

            <button

              onClick={() => {

                toast.info('Project deletion canceled.', {

                  position: 'top-center',

                  autoClose: 3000, // Keeps the toast open

                  closeButton: false,

                  draggable: false,

                  theme: 'light',

                });

                toast.dismiss(toastId,{

                  position: 'top-center',

                  autoClose: 3000, // Keeps the toast open

                  closeButton: false,

                  draggable: false,

                  theme: 'light',

                }); // Close the toast after canceling

              }}

              style={{  padding: '5px 10px', backgroundColor: '#9D0E0F', color: 'white' , border: 'none',borderRadius: '5px'}}

            >

              Cancel

            </button>

          </div>

        </div>

      ),

      autoClose: 3000, // Prevent auto-close

      closeButton: false,

      draggable: false,

      theme: 'light',

    });

  };

  // Extract file name from the full path

  const getFileName = (filePath) => {

    if (typeof filePath === 'string') {

      return filePath.split('\\').pop().split('/').pop();

    }

    if (Array.isArray(filePath)) {

      return filePath.map(file => getFileName(file)).join(', ');

    }

    return 'No file available';

  };

  return (

    <div className="projprev-overlay">

      <div className="projprev-content">

        <div className="projprevheader">

          {project.thumbnail ? (

            <img

              src={project.thumbnail.startsWith('/') ? `http://localhost:3001${project.thumbnail}` : project.thumbnail}

              alt="Thumbnail"

              className="project-thumbnail"

            />

          ) : (

            <div className="placeholder-thumbnail">No Thumbnail</div>

          )}

        </div>

        <div className="projprevcontentmain">

          <div className="projprev-left">

            <h2><strong>{project.projectName}</strong></h2>

            <p className='projdesccontainer'><strong>{project.description}</strong></p>

            {project.projectUrl && (

              <p>

                <strong>Project URL: </strong>

                <a

                  href={project.projectUrl}

                  target="\_blank"

                  rel="noopener noreferrer"

                >

                  {project.projectUrl}

                </a>

              </p>

            )}

          </div>

          <div className="projprev-right">

            <div className="projprevtags">

              {project.tag ? (

                <div className="tag-item">{project.tag}</div>

              ) : (

                <p>No tag available</p>

              )}

              {project.tools && project.tools.length > 0 ? (

                project.tools.map((tool, index) => (

                  <div key={index} className="tool-item">{tool}</div>

                ))

              ) : (

                <p>No tools available</p>

              )}

              {project.roles && project.roles.length > 0 ? (

                project.roles.map((role, index) => (

                  <div key={index} className="role-item">{role}</div>

                ))

              ) : (

                <p>No roles specified.</p>

              )}

            </div>

            <div className="projpreviewfiles">

              <p><strong>{getFileName(project.files)}</strong></p>

            </div>

          </div>

        </div>

        <div className="div">

          <button className="close-btn" onClick={onClose}>Close</button>

        </div>

      </div>

      {/\* Conditional rendering of ProjectAssessmentModal \*/}

      {showAssessmentModal && <ProjectAssessmentModal show={showAssessmentModal} onClose={() => setShowAssessmentModal(false)} />}

    </div>

  );

}

export default ProjectPreviewModal;

**Index.js:**

  app.post(

    "/api/uploadProject",

    verifyToken,

    upload.fields([

      { name: "thumbnail", maxCount: 1 },

      { name: "selectedFiles", maxCount: 10 },

    ]),

    async (req, res) => {

      try {

        const userId = req.user.id;

        const { projectName, description, tag, tools, projectUrl, assessment,roles } = req.body;

        // Validate required fields

        if (!projectName || !projectName.trim()) {

          return res.status(400).json({ success: false, message: "Project name is required." });

        }

        if (!description || !description.trim()) {

          return res.status(400).json({ success: false, message: "Description is required." });

        }

        if (!tag || !tag.trim()) {

          return res.status(400).json({ success: false, message: "A single tag is required." });

        }

        // Ensure tools is always an array

        const toolsArray = Array.isArray(tools) ? tools : tools ? [tools] : [];

        const rolesArray = Array.isArray(roles) ? roles : roles ? [roles] : [];

              // Validate roles

      if (!rolesArray.length) {

        return res.status(400).json({ success: false, message: "At least one role must be selected." });

      }

        // Parse assessment data

        const parsedAssessment = assessment

          ? JSON.parse(assessment).map((q) => ({

              ...q,

              weightedScore: q.scoring[q.rating],

            }))

          : [];

        // Validate assessment data for required categories (tags and tools)

        const requiredCategories = [

          { type: "tag", name: tag },

          ...toolsArray.map((tool) => ({ type: "tool", name: tool })),

        ];

        for (const category of requiredCategories) {

          const relevantAssessment = parsedAssessment.filter(

            (a) => a.category === category.type && a.categoryName === category.name

          );

          if (!relevantAssessment.length) {

            return res.status(400).json({

              success: false,

              message: `Assessment is required for ${category.type} '${category.name}'.`,

            });

          }

          const isValidAssessment = relevantAssessment.every(

            (item) => item.question && item.rating >= 1 && item.rating <= 5

          );

          if (!isValidAssessment) {

            return res.status(400).json({

              success: false,

              message: `Invalid assessment data for ${category.type} '${category.name}'. Ratings must be between 1 and 5.`,

            });

          }

        }

        // Retrieve files from multer

        const thumbnail = req.files["thumbnail"]

          ? `/uploads/${req.files["thumbnail"][0].filename}`

          : null;

        const selectedFiles = req.files["selectedFiles"]

          ? req.files["selectedFiles"].map((file) => file.path)

          : [];

        // Create a new project document

        const newProject = new Project({

          projectName,

          description,

          tag,

          tools: toolsArray,

          selectedFiles,

          thumbnail,

          projectUrl,

          roles: rolesArray,

          status: "pending",

          assessment: parsedAssessment,

        });

        // Save project to the database

        const savedProject = await newProject.save();

        // Associate the project with the user

        const user = await Tupath\_usersModel.findById(userId);

        if (!user) {

          return res.status(404).json({ success: false, message: "User not found" });

        }

        user.profileDetails.projects.push(savedProject.\_id);

        // Recalculate the best tag and cumulative scores

        await user.calculateBestTag();

        // Save the updated user

        await user.save();

        res.status(201).json({

          success: true,

          message: "Project uploaded successfully",

          project: savedProject,

        });

      } catch (error) {

        console.error("Error uploading project:", error);

        res.status(500).json({ success: false, message: "Internal server error" });

      }

    }

  );

  app.get("/api/projects", verifyToken, async (req, res) => {

    try {

      const userId = req.user.id;

      // Fetch user with populated projects

      const user = await Tupath\_usersModel.findById(userId).populate("profileDetails.projects");

      if (!user) {

        return res.status(404).json({ success: false, message: "User not found" });

      }

      // Add scores and tag summary for each project

      const projectsWithScores = user.profileDetails.projects.map((project) => {

        const totalScore = project.assessment.reduce((sum, question) => sum + (question.weightedScore || 0), 0);

        return {

          \_id: project.\_id,

          projectName: project.projectName,

          description: project.description,

          tag: project.tag,

          totalScore, // Sum of all weighted scores for the project

          tools: project.tools,

          status: project.status,

          assessment: project.assessment, // Include detailed assessment

          createdAt: project.createdAt,

          roles:project.roles,

        };

      });