```
const Canvas = require("../models/CanvasModel"); const Project = require("../models/ProjectModel"); const User = require("../models/UserModel"); const
Component = require("../models/ComponentModel"); const ejs = require("ejs"); const path = require("path"); const pdf = require("html-pdf") const createProject
= (newProject) => { return new Promise(async (resolve, reject) => { try { const createdCanvas = await Canvas.create({ background: "#ffffff", componentArray:
[], }); const createdProject = await Project.create({ canvasArray: [createdCanvas], projectName: newProject.projectName, owner: newProject.owner,
editorArray: [], height: newProject.height, width: newProject.width, }); if (createdProject) { resolve({ status: "OK", message: "Create success", data:
createdProject, }); } else { throw new Error("Project creation failed."); } } catch (error) { console.error("Error creating project:", error.message); reject({ status:
"ERROR", message: "Failed to create Project", error: error.message, }); } }); }; const getDetailProject = (projectId) => { return new Promise(async (resolve,
reject) => { try { const project = await Project.findOne({ _id: projectId, }).populate({ path: "canvasArray", populate: { path: "componentArray", // Trỏ tới mảng
componentArray model: "Component", // Tên model được tham chiếu }, }); if (!project) { resolve({ status: "ERROR", message: "Account is not defined!", });
return; } resolve({ status: "OK", message: "SUCCESS", data: project, }); } catch (error) { reject({ status: "ERROR", message: "Failed to create Project", error:
error.message, }); } }); }; const getAllProject = (userId) => { return new Promise(async (resolve, reject) => { try { const projects = await Project.find({ owner:
userId, copy: null }).populate({ path: "canvasArray", populate: { path: "componentArray", // Trỏ tới mảng componentArray model: "Component", // Tên model
được tham chiếu }, }); if (!projects) { resolve({ status: "ERROR", message: "Account is not defined!", }); return; } resolve({ status: "OK", message: "SUCCESS",
data: projects, }); } catch (error) { reject({ status: "ERROR", message: "Failed to create Project", error: error.message, }); } }); }; const getAllTeamProject =
(userId) => { return new Promise(async (resolve, reject) => { try { const user = await User.findById(userId) const projects = await Project.find({ editorArray: {
$in: [user.email] }, }).populate({ path: "canvasArray", populate: { path: "componentArray", // Trỏ tới mảng componentArray model: "Component", // Tên model
được tham chiếu }, }); if (!projects) { resolve({ status: "ERROR", message: "Account is not defined!", }); return; } resolve({ status: "OK", message: "SUCCESS",
data: projects, }); } catch (error) { reject({ status: "ERROR", message: "Failed to create Project", error: error.message, }); } }); }; const getPublic = () => {
return new Promise(async (resolve, reject) => { try { const projects = await Project.find({ isPublic: true, copy: { $ne: null }, }).populate({ path: "canvasArray",
populate: { path: "componentArray", // Trỏ tới mảng componentArray model: "Component", // Tên model được tham chiếu }, }); if (!projects) { resolve({ status:
"ERROR", message: "Account is not defined!", }); return; } resolve({ status: "OK", message: "SUCCESS", data: projects, }); } catch (error) { reject({ status:
"ERROR", message: "Failed to create Project", error: error.message, }); } }); }; const updateProject = (projectId, data) => { return new Promise(async (resolve,
reject) => { try { console.log("data", data) const checkProject = await Project.findOne({ _id: projectId, }).populate({ path: "canvasArray", populate: { path:
"componentArray", // Trỏ tới mảng componentArray model: "Component", // Tên model được tham chiếu }, }); if (!checkProject) { resolve({ status: "ERROR",
message: "Project is not defined!", }); return; } const updatedCanvasCopy = JSON.parse(JSON.stringify(data)); delete updatedCanvasCopy._id
updatedCanvasCopy.componentArray = await Promise.all( (updatedCanvasCopy.componentArray || []).map(async (component) => { const newComponent = {
...component }; delete newComponent._id; // Lưu component mới vào database const savedComponent = await Component.create(newComponent); return
savedComponent; // Trả về toàn bộ thông tin hoặc chỉ `_id` }) ); // Lưu canvas copy mới vào database const savedCanvasCopy = await
Canvas.create(updatedCanvasCopy); let createdCanvas = null if (data && Object.keys(data).length > 0) { createdCanvas = savedCanvasCopy } else {
createdCanvas = await Canvas.create({ background: "#ffffff", componentArray: [], }); } const canvasArray = [...checkProject.canvasArray, createdCanvas._id];
const updatedProject = await Project.findByIdAndUpdate( projectId, { canvasArray }, { new: true } ); if (!updatedProject) { resolve({ status: "ERROR",
message: "Project update failed or not found", }); return; } resolve({ status: "OK", message: "SUCCESS", data: updatedProject, }); } catch (error) { reject({
status: "ERROR", message: "Failed to update Project", error: error.message, }); } }); }; const addProject = (projectId, data) => { return new Promise(async
(resolve, reject) => { try { console.log("data", data.canvasArray) const checkProject = await Project.findOne({ _id: projectId, }).populate({ path: "canvasArray",
populate: { path: "componentArray", // Trỏ tới mảng componentArray model: "Component", // Tên model được tham chiếu }, }); if (!checkProject) { resolve({
status: "ERROR", message: "Project is not defined!", }); return; } const newCanvases = await Promise.all( (data.canvasArray || []).map(async (canvas) => { //
Tạo một bản sao canvas và xử lý componentArray const updatedCanvasCopy = JSON.parse(JSON.stringify(canvas)); delete updatedCanvasCopy._id;
updatedCanvasCopy.componentArray = await Promise.all( (updatedCanvasCopy.componentArray || []).map(async (component) => { const newComponent = {
...component }; delete newComponent._id; // Lưu component mới vào database const savedComponent = await Component.create(newComponent); return
savedComponent; // Trả về toàn bộ thông tin hoặc chỉ `_id` }) ); // Lưu canvas mới vào database return await Canvas.create(updatedCanvasCopy); }) ); // Cập
nhật mảng canvasArray của project const canvasArray = [ ...(checkProject.canvasArray || []).map((canvas) => canvas._id), ...newCanvases.map((canvas) =>
canvas._id), ]; console.log('canvasArray', canvasArray) const updatedProject = await Project.findByIdAndUpdate( projectId, { canvasArray }, { new: true }
).populate({ path: "canvasArray", populate: { path: "componentArray", model: "Component", }, }); if (!updatedProject) { resolve({ status: "ERROR", message:
"Project update failed or not found", }); return; } resolve({ status: "OK", message: "SUCCESS", data: updatedProject, }); } catch (error) { reject({ status:
"ERROR", message: "Failed to update Project", error: error.message, }); } }); }; const updatePublic = async (projectId) => { try { const checkProject = await
Project.findOne({ _id: projectId }).populate({ path: "canvasArray", populate: { path: "componentArray", model: "Component", }, }); checkProject.isPublic = true
checkProject.save(); if (!checkProject) { return { status: "ERROR", message: "Project is not defined!", }; } const updatedProjectCopy =
JSON.parse(JSON.stringify(checkProject)); updatedProjectCopy.copy = projectId // Loại bỏ `_id` của project để MongoDB tự sinh mới delete
updatedProjectCopy._id; updatedProjectCopy.isPublic = true; // Tạo và lưu từng `canvas` mới updatedProjectCopy.canvasArray = await Promise.all(
(updatedProjectCopy.canvasArray || []).map(async (canvas) => { const newCanvas = { ...canvas }; delete newCanvas._id; newCanvas.componentArray = await
Promise.all( (canvas.componentArray || []).map(async (component) => { const newComponent = { ...component }; delete newComponent._id; // Lưu
component mới vào database const savedComponent = await Component.create(newComponent); return savedComponent; // Trả về toàn bộ thông tin hoặc chỉ
`_id` }) ); // Lưu canvas mới vào database const savedCanvas = await Canvas.create(newCanvas); return savedCanvas; // Trả về toàn bộ thông tin hoặc chỉ `_id`
}) ); // Lưu project mới vào database const savedProject = await Project.create(updatedProjectCopy); const populatedProject = await
Project.findById(savedProject._id).populate({ path: "canvasArray", populate: { path: "componentArray", model: "Component", }, }); return { status: "OK",
message: "SUCCESS", data: populatedProject, }; } catch (error) { return { status: "ERROR", message: "Failed to update Project", error: error.message, }; } };
const updatePrivate = async (projectId) => { try { const checkProject = await Project.findOne({ _id: projectId }).populate({ path: "canvasArray", populate: {
path: "componentArray", model: "Component", }, }); checkProject.isPublic = false checkProject.save(); await Project.findOneAndDelete({copy: projectId})
return { status: "OK", message: "Private project success", }; } catch (error) { return { status: "ERROR", message: "Failed to update Project", error: error.message,
}; } }; const deleteProject = (projectId) => { return new Promise(async (resolve, reject) => { try { const project = await Project.findOne({ _id: projectId, }); if
(!project) { resolve({ status: "ERROR", message: "Project is not defined!", }); return; } await Project.findByIdAndDelete(projectId); resolve({ status: "OK",
message: "Delete success", }); } catch (error) { reject({ status: "ERROR", message: "Failed to create Project", error: error.message, }); } }); }; const addEditor =
async (projectId, email) => { try { const project = await Project.findById(projectId); if (!project) { return { status: "ERROR", message: "Project not found!", }; }
if (project.editors.includes(email)) { return { status: "ERROR", message: "Editor is already added!", }; } project.editors.push(email); const updatedProject = await
project.save(); return { status: "OK", message: "SUCCESS", data: updatedProject, }; } catch (error) { return { status: "ERROR", message: "Failed to update
Project", error: error.stack, }; } }; const downloadProject = async (projectId, data) => { try { const project = await Project.findById(projectId).populate({ path:
"canvasArray", populate: { path: "componentArray", model: "Component", }, }); if (!project) { return { status: "ERROR", message: "Project not found!", }; }
const templatePath = path.resolve("D:/ĐỒ ÁN 1/Chung/sparkle/backend/services/ProjectService.js"); const htmlContent = await ejs.renderFile(templatePath, {
project }); const pdfOptions = { format: "A4", orientation: "portrait" }; const pdfPath = path.join(__dirname, `documents/${project.projectName}.pdf`);
pdf.create(htmlContent, pdfOptions).toFile(pdfPath, (err) => { if (err) { return res.status(500).json({ status: "ERROR", message: "PDF generation failed", error:
err.message }); } // Gửi file PDF về client res.download(pdfPath, `${project.projectName}.pdf`); }); } catch (error) { return { status: "ERROR", message:
"Failed to download Project", error: error.message, }; } }; module.exports = { createProject, getDetailProject, getAllProject, getAllTeamProject, getPublic,
updateProject, addProject, updatePublic, updatePrivate, deleteProject, addEditor, downloadProject, };
```