React Style Guide

General Rules/Procedures

If an object is re-used many times, make it a component. For example, make a special component for the buttons since we use them everywhere. Modals should also be in their own components so that their handlers/states are more clearly separated from the page they are rendered on.

If you want to cut down App.css, I would recommend to utilize classes more for styling and also maybe split off component-specific styling to a dedicated .css file.

Always import Navigate from react-router-dom and use it whenever you want to change pages. Do not use window.location.href because it messes with React's rendering tree. Navigate cleanly negotiates around the rendering tree, so it will cause fewer errors.

UseMemo() is used to build arrays of content and render it in React. Anytime you want to render an array, do it with UseMemo. It makes rendering faster and prevents react from recalculating the arrays on every render.

Handlers should be written with the useCallback() function, which prevents unnecessary re-rendering, which will also make our pages faster.

Always double-check that your dependencies are correct. Dependency issues cause infinite reruns.

Rule-of-thumb is if you want the page to do something that we've never previously programmed a page to do before, check if React has a way to do it. The answer is often yes, and it's better to follow how React wants it done than to try to code up a solution yourself from scratch, then struggle with React when your solution breaks it somehow.

Dashboard Example

Here's the dashboard.jsx page with these changes incorporated as produced by ChatGPT. It's not tested, so don't trust it entirely, but use it as an example of how to write these functions.

```
import React, { useState, useEffect, useMemo, useCallback } from 'react';
```

```
import { useParams, Navigate } from 'react-router-dom';
import { useAuth } from '../AuthProvider';
import DashboardNav from '../components/DashboardNav';
import ResourceCard from '../components/ResourceCard';

const ClassroomPage = () => {
    //=================== Extract Auth ==================//
    const {
        userdata, token,
        classrooms, addUser, removeUser, isMember,
        deleteClassroom, loadContent,
        createQuiz, createFlashcard
    } = useAuth();

    const { classroomId } = useParams();

    //=================== State ==================//
    const [activeTab, setActiveTab] = useState('content');
    const [creationStep, setCreationStep] = useState(null);
    const [content, setContent] = useState([]);
    const [isUserJoined, setIsUserJoined] = useState(false);
    const [newContentData, setNewContentData] = useState({
        name: '',
        description: '',
    });

    //=================== Derived Values ==================//
    const currentClassroom = useMemo(
        () => classrooms.find((r) => String(r.id) === classroomId),
        [classrooms, classroomId]
    );

    const modalTitle = useMemo(() => {
        switch (creationStep) {
            case 'selectType': return 'Select Content Type';
            case 'flashcard': return 'Create Flash Card Set';
            case 'quiz': return 'Create Quiz';
            default: return '';
        }
    }, [creationStep]);

    //=================== Guards ==================//
    if (!token || !userdata?.id) return <Navigate to="/" replace />;
    if (!currentClassroom) return <Navigate to="/dashboard" replace />;

    //=================== Load content ==================//
    useEffect(() => {
        loadContent(userdata, classroomId, setContent);
    }, [userdata, classroomId, loadContent]);

    //=================== Initialize membership ==================//
    useEffect(() => {
        const loadMembership = async () => {
```

```
        const member = await isMember(currentClassroom.id);
        setIsUserJoined(member);
    };

    loadMembership();
}, [currentClassroom.id, isMember]);

//=================== Handlers ===================//
const handleAddTypeClick = useCallback(() => {
    setCreationStep('selectType');
}, []);

const closeCreationModal = useCallback(() => {
    setCreationStep(null);
    setNewContentData({ name: '', description: '' });
}, []);

const handleSelectType = useCallback((type) => {
    setCreationStep(type);
}, []);

const handleFormChange = useCallback((e) => {
    const { name, value } = e.target;
    setNewContentData((prev) => ({ ...prev, [name]: value }));
}, []);

const handleJoinUser = useCallback(() => {
    if (isUserJoined) return alert("You are already a member.");
    addUser(currentClassroom.id);
    setIsUserJoined(true);
}, [isUserJoined, addUser, currentClassroom?.id]);

const handleLeaveUser = useCallback(() => {
    if (!isUserJoined) return alert("You are not a member.");
    removeUser(currentClassroom.id);
    setIsUserJoined(false);
}, [isUserJoined, removeUser, currentClassroom?.id]);

const handleDeleteClassroom = useCallback(async () => {
    if (userdata.id !== currentClassroom.ownerId)
        return alert("You cannot delete this classroom.");

    await deleteClassroom(currentClassroom.id);
}, [userdata?.id, currentClassroom, deleteClassroom]);

const handleCreateContentSubmit = useCallback(async (e) => {
    e.preventDefault();

    if (creationStep === "quiz") {
        const payload = {
            title: newContentData.name,
            description: newContentData.description,
            classRoomId: classroomId,
```

```
            creatorId: userdata.id
        };

        const savedQuiz = await createQuiz(payload);
        if (savedQuiz) {
            setContent((prev) => [
                ...prev,
                {
                    id: savedQuiz.Id,
                    name: savedQuiz.Title,
                    type: "Quiz",
                    summary: savedQuiz.Description || "No description"
                }
            ]);
        }
    }

    if (creationStep === "flashcard") {
        const payload = {
            classRoomId: classroomId,
            title: newContentData.name,
            information: newContentData.description,
            tags: "default"
        };

        const savedFC = await createFlashcard(payload);
        if (savedFC) {
            setContent((prev) => [
                ...prev,
                {
                    id: savedFC.id,
                    name: newContentData.name,
                    type: "Flashcard",
                    summary: newContentData.description || "Flashcard"
                }
            ]);
        }
    }

    closeCreationModal();
}, [
    creationStep,
    newContentData,
    createQuiz,
    createFlashcard,
    classroomId,
    userdata?.id,
    closeCreationModal
]);

//=================== Modal Renderer ===================//
const renderCreationModalContent = useCallback(() => {
    if (creationStep === "selectType") {
```

```
            return (
                <>
                    <p className="add-type-text">Choose a content type:</p>
                    <div className="add-type-options">
                        <button className="add-type-card" onClick={() =>
handleSelectType("flashcard")}>
                            <h4>Flashcard</h4>
                            <p>Definitions, concepts, key terms.</p>
                        </button>
                        <button className="add-type-card" onClick={() =>
handleSelectType("quiz")}>
                            <h4>Quiz</h4>
                            <p>Create assessment questions.</p>
                        </button>
                    </div>
                </>
            );
        }

        if (creationStep === "quiz" || creationStep === "flashcard") {
            const typeName = creationStep === "quiz" ? "Quiz" : "Flashcard Set";

            return (
                <form onSubmit={handleCreateContentSubmit}>
                    <label>{typeName} Name:</label>
                    <input
                        type="text"
                        name="name"
                        value={newContentData.name}
                        onChange={handleFormChange}
                        required
                        className="form-input-text"
                    />

                    <label>Description:</label>
                    <textarea
                        name="description"
                        value={newContentData.description}
                        onChange={handleFormChange}
                        rows="2"
                        maxLength="150"
                        className="form-input-text"
                    />

                    <button
                        type="submit"
                        disabled={!newContentData.name}
                        className="dashboard-btn form-submit-btn"
                    >
                        Create {typeName}
                    </button>
                </form>
            );
```

```
        }

        return null;
    }, [creationStep, newContentData, handleFormChange, handleSelectType,
handleCreateContentSubmit]);

    //=================== JSX UI ===================//
    return (
        <main className="dashboard-page">
            <DashboardNav initialActiveTab="content" onTabChange={setActiveTab} />

            {/* Join / Leave Buttons */}
            <section className="classroom-button-box">
                <button onClick={handleJoinUser}>Join Classroom</button>
                <button onClick={handleLeaveUser}>Leave Classroom</button>
            </section>

            {/* Delete Button */}
            <section className="delete-button-box">
                <button onClick={handleDeleteClassroom}>Delete Classroom</button>
            </section>

            {/* Content Section */}
            {(activeTab === "content" || activeTab === "classrooms") && (
                <section className="dashboard-box">
                    <div className="dashboard-box-header">
                        <h2>{currentClassroom.name} Content</h2>
                    </div>

                    {content.length === 0 ? (
                        <p>No content yet. Click + to add some!</p>
                    ) : (
                        <div className="classroom-grid">
                            {content.map((item) => (
                                <ResourceCard key={item.id} resource={item}
isClassroomLevel={false} />
                            ))}
                        </div>
                    )}

                    <button className="floating-add-btn"
onClick={handleAddTypeClick}>+</button>

                    {creationStep && (
                        <div className="add-type-overlay" onClick={closeCreationModal}>
                            <div className="add-type-modal" onClick={(e) =>
e.stopPropagation()}>
                                <div className="add-type-header">
                                    <h3>{modalTitle}</h3>
                                    <button className="add-type-close"
onClick={closeCreationModal}>x</button>
                                </div>
                                {renderCreationModalContent()}
```

```
                            </div>
                        </div>
                    )}
                </section>
            )}

            {/* Profile Section */}
            {activeTab === "profile" && (
                <section className="dashboard-box">
                    <h2>Profile</h2>
                    <p><strong>Username:</strong> {userdata?.name}</p>
                    <p><strong>Email:</strong> {userdata?.email}</p>
                    <p><strong>Role:</strong> Student</p>
                </section>
            )}
        </main>
    );
};

export default ClassroomPage;
```