```
import React, { useState, useEffect, createContext, useContext,
useCallback } from 'react';
import { initializeApp } from 'firebase/app';
import { getAuth, signInAnonymously, signInWithCustomToken,
onAuthStateChanged } from 'firebase/auth';
import { getFirestore, collection, addDoc, getDocs, doc, updateDoc,
deleteDoc, query, where, onSnapshot } from 'firebase/firestore';
import { getStorage, ref as storageRef, uploadBytesResumable,
getDownloadURL, deleteObject } from 'firebase/storage';

// Define API key (empty string for Canvas runtime auto-provisioning)
const API_KEY = "";

// Create a context for Firebase services and user data
const FirebaseContext = createContext(null);

// Custom hook to use Firebase context
const useFirebase = () => {
  return useContext(FirebaseContext);
};

// Removed AnimateScaleIn component and its associated styles.
// Modals will appear directly without the custom scaling animation.

// --- Modals and Utility Components ---

/**
 * @param {object} props
 * @param {string} props.message
 * @param {Function} props.onClose
 */
const InfoModal = ({ message, onClose }) => (
  <div className="fixed inset-0 bg-black bg-opacity-50 flex
items-center justify-center p-4 z-50">
    <div className="bg-white rounded-lg p-6 shadow-xl max-w-sm w-full
transition-all duration-300">
      <p className="text-gray-800 text-center text-lg
mb-6">{message}</p>
      <button
        onClick={onClose}
        className="w-full bg-blue-600 hover:bg-blue-700 text-white
font-bold py-2 px-4 rounded-lg transition duration-200 ease-in-out
focus:outline-none focus:ring-2 focus:ring-blue-500
focus:ring-opacity-75"
      >
        Close
      </button>
    </div>
```

```
  </div>
);

/**
 * @param {object} props
 * @param {string} props.message
 * @param {Function} props.onConfirm
 * @param {Function} props.onCancel
 */
const ConfirmModal = ({ message, onConfirm, onCancel }) => (
  <div className="fixed inset-0 bg-black bg-opacity-50 flex
items-center justify-center p-4 z-50">
    <div className="bg-white rounded-lg p-6 shadow-xl max-w-sm w-full
transition-all duration-300">
      <p className="text-gray-800 text-center text-lg
mb-6">{message}</p>
      <div className="flex justify-around space-x-4">
        <button
          onClick={onConfirm}
          className="flex-1 bg-red-600 hover:bg-red-700 text-white
font-bold py-2 px-4 rounded-lg transition duration-200 ease-in-out
focus:outline-none focus:ring-2 focus:ring-red-500
focus:ring-opacity-75"
        >
          Confirm
        </button>
        <button
          onClick={onCancel}
          className="flex-1 bg-gray-300 hover:bg-gray-400
text-gray-800 font-bold py-2 px-4 rounded-lg transition duration-200
ease-in-out focus:outline-none focus:ring-2 focus:ring-gray-500
focus:ring-opacity-75"
        >
          Cancel
        </button>
      </div>
    </div>
  </div>
);

// --- File Upload Utility ---

/**
 * Uploads a file to Firebase Storage.
 * @param {File} file The file to upload.
 * @param {object} storage Firebase Storage instance.
 * @param {string} path The storage path (e.g., 'syllabi/courseId/',
'assignments/courseId/').
```

```
 * @param {Function} setUploadProgress State setter for upload
progress (0-100).
 * @param {Function} setInfoModalMessage State setter for info modal
message.
 * @param {Function} setShowInfoModal State setter to show info modal.
 * @returns {Promise<string|null>} A promise that resolves with the
download URL or null if upload fails.
 */
const uploadFileToFirebase = async (file, storage, path,
setUploadProgress, setInfoModalMessage, setShowInfoModal) => {
  if (!file || !storage) {
    setInfoModalMessage('No file selected for upload or storage not
ready.');
    setShowInfoModal(true);
    return null;
  }

  return new Promise((resolve) => {
    const filePath = `${path}${file.name}_${new Date().getTime()}`;
    const fileRef = storageRef(storage, filePath);
    const uploadTask = uploadBytesResumable(fileRef, file);

    uploadTask.on(
      'state_changed',
      (snapshot) => {
        const progress = (snapshot.bytesTransferred /
snapshot.totalBytes) * 100;
        setUploadProgress(progress);
      },
      (error) => {
        console.error("Upload failed:", error);
        setInfoModalMessage(`File upload failed: ${error.message}`);
        setShowInfoModal(true);
        resolve(null);
      },
      async () => {
        try {
          const downloadURL = await
getDownloadURL(uploadTask.snapshot.ref);
          setInfoModalMessage('File uploaded successfully!');
          setShowInfoModal(true);
          resolve(downloadURL);
        } catch (error) {
          console.error("Error getting download URL:", error);
          setInfoModalMessage(`Error getting file URL:
${error.message}`);
          setShowInfoModal(true);
          resolve(null);
```

```
        }
      }
    );
  });
};

/**
 * Deletes a file from Firebase Storage.
 * @param {string} fileUrl The download URL of the file to delete.
 * @param {object} storage Firebase Storage instance.
 * @param {Function} setInfoModalMessage State setter for info modal
message.
 * @param {Function} setShowInfoModal State setter to show info modal.
 * @returns {Promise<boolean>} True if deletion was successful, false
otherwise.
 */
const deleteFileFromFirebase = async (fileUrl, storage,
setInfoModalMessage, setShowInfoModal) => {
  if (!fileUrl || !storage) {
    return false;
  }
  try {
    const fileRef = storageRef(storage, fileUrl); // Create ref from
download URL
    await deleteObject(fileRef);
    setInfoModalMessage('File deleted successfully.');
    setShowInfoModal(true);
    return true;
  } catch (error) {
    console.error("Error deleting file:", error);
    setInfoModalMessage(`Error deleting file: ${error.message}`);
    setShowInfoModal(true);
    return false;
  }
};


// --- Course Management Components ---

/**
 * Form for adding/editing a course.
 * @param {object} props
 * @param {Function} props.onSave
 * @param {Function} props.onCancel
 * @param {object | null} props.initialCourseData
 */
const CourseForm = ({ onSave, onCancel, initialCourseData = null }) =>
{
```

```
  const [courseName, setCourseName] = useState(initialCourseData?.name
|| '');
  const [description, setDescription] =
useState(initialCourseData?.description || '');
  const [showInfoModal, setShowInfoModal] = useState(false);
  const [infoModalMessage, setInfoModalMessage] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    if (!courseName.trim()) {
      setInfoModalMessage('Course Name cannot be empty.');
      setShowInfoModal(true);
      return;
    }
    onSave({ name: courseName, description });
  };

  return (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex
items-center justify-center p-4 z-40">
      <div className="bg-white rounded-lg p-6 shadow-xl max-w-lg
w-full transform transition-all duration-300">
        <h2 className="text-2xl font-semibold mb-6 text-gray-800
text-center">
          {initialCourseData ? 'Edit Course' : 'Add New Course'}
        </h2>
        <form onSubmit={handleSubmit} className="space-y-4">
          <div>
            <label htmlFor="courseName" className="block text-gray-700
text-sm font-bold mb-2">
              Course Name:
            </label>
            <input
              type="text"
              id="courseName"
              value={courseName}
              onChange={(e) => setCourseName(e.target.value)}
              className="shadow appearance-none border rounded-lg
w-full py-3 px-4 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition
duration-200"
              placeholder="e.g., Algebra I"
            />
          </div>
          <div>
            <label htmlFor="description" className="block
text-gray-700 text-sm font-bold mb-2">
              Description:
```

```
            </label>
            <textarea
              id="description"
              value={description}
              onChange={(e) => setDescription(e.target.value)}
              className="shadow appearance-none border rounded-lg
w-full py-3 px-4 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition
duration-200 h-24 resize-y"
              placeholder="e.g., Introduction to basic algebraic
concepts."
            ></textarea>
          </div>
          <div className="flex justify-end space-x-4 pt-4">
            <button
              type="button"
              onClick={onCancel}
              className="bg-gray-300 hover:bg-gray-400 text-gray-800
font-bold py-2 px-5 rounded-lg transition duration-200 ease-in-out
focus:outline-none focus:ring-2 focus:ring-gray-500
focus:ring-opacity-75"
            >
              Cancel
            </button>
            <button
              type="submit"
              className="bg-blue-600 hover:bg-blue-700 text-white
font-bold py-2 px-5 rounded-lg transition duration-200 ease-in-out
focus:outline-none focus:ring-2 focus:ring-blue-500
focus:ring-opacity-75"
            >
              {initialCourseData ? 'Save Changes' : 'Add Course'}
            </button>
          </div>
        </form>
      </div>
      {showInfoModal && <InfoModal message={infoModalMessage}
onClose={() => setShowInfoModal(false)} />}
    </div>
  );
};

/**
 * Displays a list of courses and allows adding/editing/deleting them.
 * @param {object} props
 * @param {Function} props.onSelectCourse
 */
const CourseList = ({ onSelectCourse }) => {
```

```
  const { db, userId, isAuthReady } = useFirebase();
  const [courses, setCourses] = useState([]);
  const [showAddCourseForm, setShowAddCourseForm] = useState(false);
  const [editingCourse, setEditingCourse] = useState(null);
  const [showConfirmModal, setShowConfirmModal] = useState(false);
  const [courseToDelete, setCourseToDelete] = useState(null);
  const [showInfoModal, setShowInfoModal] = useState(false);
  const [infoModalMessage, setInfoModalMessage] = useState('');

  // Default courses to add if list is empty
  const defaultCourses = [
    { name: 'Composition 1', description: 'Fundamental principles of
academic writing and rhetoric.' },
    { name: 'Composition 2', description: 'Advanced research,
argumentation, and literary analysis.' },
  ];

  // Function to add default courses
  const addDefaultCoursesToDb = useCallback(async () => {
    if (!db || !userId) {
      console.warn("Database not ready for adding default courses.");
      return;
    }
    const coursesCollectionRef = collection(db,
`artifacts/${__app_id}/users/${userId}/courses`);
    const existingCoursesSnapshot = await
getDocs(coursesCollectionRef);
    if (existingCoursesSnapshot.empty) { // Only add if no courses
exist
      console.log("Adding default courses...");
      for (const course of defaultCourses) {
        try {
          await addDoc(coursesCollectionRef, course);
        } catch (e) {
          console.error("Error adding default course:", e);
        }
      }
      setInfoModalMessage('Default courses (Composition 1 & 2)
added!');
      setShowInfoModal(true);
    }
  }, [db, userId]);

  // Fetch courses from Firestore and add defaults if needed
  useEffect(() => {
    if (!isAuthReady || !db || !userId) {
      return;
    }
```

```
    const coursesCollectionRef = collection(db,
`artifacts/${__app_id}/users/${userId}/courses`);
    const q = query(coursesCollectionRef);

    const unsubscribe = onSnapshot(q, (snapshot) => {
      const coursesData = snapshot.docs.map(doc => ({
        id: doc.id,
        ...doc.data()
      }));
      setCourses(coursesData);

      // If no courses are found after initial fetch, add defaults
      if (coursesData.length === 0) {
        addDefaultCoursesToDb();
      }
    }, (error) => {
      console.error("Error fetching courses:", error);
      setInfoModalMessage('Failed to load courses. Please try
again.');
      setShowInfoModal(true);
    });

    return () => unsubscribe();
  }, [db, userId, isAuthReady, addDefaultCoursesToDb]); // Include
addDefaultCoursesToDb in dependencies

  // Handle adding a new course
  const handleAddCourse = async (courseData) => {
    if (!db || !userId) {
      setInfoModalMessage('Database not ready. Please wait.');
      setShowInfoModal(true);
      return;
    }
    try {
      await addDoc(collection(db,
`artifacts/${__app_id}/users/${userId}/courses`), courseData);
      setInfoModalMessage('Course added successfully!');
      setShowInfoModal(true);
      setShowAddCourseForm(false);
    } catch (e) {
      console.error("Error adding document: ", e);
      setInfoModalMessage('Error adding course. Please try again.');
      setShowInfoModal(true);
    }
  };

  // Handle updating an existing course
```

```
  const handleUpdateCourse = async (courseData) => {
    if (!db || !userId || !editingCourse) {
      setInfoModalMessage('Database or course data not ready. Please
wait.');
      setShowInfoModal(true);
      return;
    }
    try {
      const courseRef = doc(db,
`artifacts/${__app_id}/users/${userId}/courses`, editingCourse.id);
      await updateDoc(courseRef, courseData);
      setInfoModalMessage('Course updated successfully!');
      setShowInfoModal(true);
      setEditingCourse(null);
    } catch (e) {
      console.error("Error updating document: ", e);
      setInfoModalMessage('Error updating course. Please try again.');
      setShowInfoModal(true);
    }
  };

  // Prepare for course deletion
  const confirmDeleteCourse = (course) => {
    setCourseToDelete(course);
    setShowConfirmModal(true);
  };

  // Handle deleting a course
  const handleDeleteCourse = async () => {
    if (!db || !userId || !courseToDelete) {
      setInfoModalMessage('Database or course data not ready. Please
wait.');
      setShowInfoModal(true);
      return;
    }
    try {
      // First, delete related assignments, syllabi, and student
samples
      const collectionsToDelete = ['assignments', 'syllabi',
'studentSamples'];
      for (const collectionName of collectionsToDelete) {
        const itemsRef = collection(db,
`artifacts/${__app_id}/users/${userId}/${collectionName}`);
        const itemsQuery = query(itemsRef, where("courseId", "==",
courseToDelete.id));
        const itemsSnapshot = await getDocs(itemsQuery);
        for (const doc of itemsSnapshot.docs) {
          const data = doc.data();
```

```
          // Delete associated files if they exist
          if (data.fileUrl &&
data.fileUrl.startsWith('https://firebasestorage.googleapis.com/')) {
            await deleteFileFromFirebase(data.fileUrl,
useFirebase().storage, setInfoModalMessage, setShowInfoModal);
          }
          if (data.instructionsFileUrl &&
data.instructionsFileUrl.startsWith('https://firebasestorage.googleapi
s.com/')) {
            await deleteFileFromFirebase(data.instructionsFileUrl,
useFirebase().storage, setInfoModalMessage, setShowInfoModal);
          }
          if (data.rubricFileUrl &&
data.rubricFileUrl.startsWith('https://firebasestorage.googleapis.com/
')) {
            await deleteFileFromFirebase(data.rubricFileUrl,
useFirebase().storage, setInfoModalMessage, setShowInfoModal);
          }
          if (data.sampleFileUrl &&
data.sampleFileUrl.startsWith('https://firebasestorage.googleapis.com/
')) {
            await deleteFileFromFirebase(data.sampleFileUrl,
useFirebase().storage, setInfoModalMessage, setShowInfoModal);
          }
          await deleteDoc(doc.ref);
        }
      }

      // Finally, delete the course itself
      const courseRef = doc(db,
`artifacts/${__app_id}/users/${userId}/courses`, courseToDelete.id);
      await deleteDoc(courseRef);

      setInfoModalMessage('Course and related data deleted
successfully!');
      setShowInfoModal(true);
    } catch (e) {
      console.error("Error deleting course: ", e);
      setInfoModalMessage('Error deleting course. Please try again.');
      setShowInfoModal(true);
    } finally {
      setShowConfirmModal(false);
      setCourseToDelete(null);
    }
  };

  return (
    <div className="p-4 sm:p-6 lg:p-8 flex-grow">
```

```
      <h1 className="text-3xl sm:text-4xl font-bold text-gray-800 mb-6
text-center">Your Courses</h1>

      <div className="flex justify-center mb-6">
        <button
          onClick={() => setShowAddCourseForm(true)}
          className="bg-green-600 hover:bg-green-700 text-white
font-bold py-3 px-6 rounded-lg shadow-md transition duration-200
ease-in-out transform hover:scale-105 focus:outline-none focus:ring-2
focus:ring-green-500 focus:ring-opacity-75"
        >
          Add New Course
        </button>
      </div>

      {courses.length === 0 ? (
        <p className="text-center text-gray-600 text-lg mt-8">No
courses added yet. Click "Add New Course" to get started! (Composition
1 & 2 will auto-add soon if database is empty)</p>
      ) : (
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3
gap-6">
          {courses.map(course => (
            <div
              key={course.id}
              className="bg-white rounded-lg shadow-lg p-6 transform
transition duration-300 hover:scale-105 border border-gray-200 flex
flex-col"
            >
              <h2 className="text-xl font-semibold text-gray-800
mb-2">{course.name}</h2>
              <p className="text-gray-600 text-sm mb-4
flex-grow">{course.description || 'No description provided.'}</p>
              <div className="flex justify-end space-x-3 mt-auto">
                <button
                  onClick={() => onSelectCourse(course)}
                  className="bg-blue-500 hover:bg-blue-600 text-white
text-sm font-medium py-2 px-4 rounded-lg transition duration-200
focus:outline-none focus:ring-2 focus:ring-blue-500"
                >
                  View Details
                </button>
                <button
                  onClick={() => setEditingCourse(course)}
                  className="bg-yellow-500 hover:bg-yellow-600
text-white text-sm font-medium py-2 px-4 rounded-lg transition
duration-200 focus:outline-none focus:ring-2 focus:ring-yellow-500"
                >
```

```
                  Edit
                </button>
                <button
                  onClick={() => confirmDeleteCourse(course)}
                  className="bg-red-500 hover:bg-red-600 text-white
text-sm font-medium py-2 px-4 rounded-lg transition duration-200
focus:outline-none focus:ring-2 focus:ring-red-500"
                >
                  Delete
                </button>
              </div>
            </div>
          ))}
        </div>
      )}

      {showAddCourseForm && (
        <CourseForm
          onSave={handleAddCourse}
          onCancel={() => setShowAddCourseForm(false)}
        />
      )}

      {editingCourse && (
        <CourseForm
          initialCourseData={editingCourse}
          onSave={handleUpdateCourse}
          onCancel={() => setEditingCourse(null)}
        />
      )}

      {showConfirmModal && (
        <ConfirmModal
          message={`Are you sure you want to delete
"${courseToDelete?.name}"? This will also delete all associated
assignments, syllabi, and student samples.`}
          onConfirm={handleDeleteCourse}
          onCancel={() => setShowConfirmModal(false)}
        />
      )}

      {showInfoModal && <InfoModal message={infoModalMessage}
onClose={() => setShowInfoModal(false)} />}
    </div>
  );
};

// --- Rubric Display Component ---
```

```
/**
 * Displays a rubric in a visually appealing table format.
 * @param {object} props
 * @param {object} props.rubric The rubric object with title,
description, and criteria.
 */
const RubricDisplay = ({ rubric }) => {
  if (!rubric || !rubric.criteria || rubric.criteria.length === 0) {
    return <p className="text-gray-600">No rubric data available.</p>;
  }

  const allLevels = Array.from(new Set(rubric.criteria.flatMap(c =>
c.levels.map(l => l.levelName))));

  return (
    <div className="bg-white p-6 rounded-lg shadow-inner border
border-gray-200">
      <h3 className="text-xl font-semibold text-gray-800
mb-2">{rubric.title || 'Rubric'}</h3>
      {rubric.description && <p className="text-gray-600 text-sm
mb-4">{rubric.description}</p>}

      <div className="overflow-x-auto">
        <table className="min-w-full divide-y divide-gray-300
rounded-lg overflow-hidden">
          <thead className="bg-gray-100">
            <tr>
              <th scope="col" className="px-4 py-2 text-left text-xs
font-medium text-gray-700 uppercase tracking-wider rounded-tl-lg">
                Criteria
              </th>
              {allLevels.map(levelName => (
                <th key={levelName} scope="col" className="px-4 py-2
text-left text-xs font-medium text-gray-700 uppercase tracking-wider">
                  {levelName}
                </th>
              ))}
            </tr>
          </thead>
          <tbody className="bg-white divide-y divide-gray-200">
            {rubric.criteria.map((criterion, index) => (
              <tr key={index} className="hover:bg-gray-50">
                <td className="px-4 py-3 whitespace-nowrap text-sm
font-medium text-gray-900 bg-gray-50">
                  {criterion.name}
                </td>
                {allLevels.map(levelName => {
                  const level = criterion.levels.find(l => l.levelName
```

```
=== levelName);
                  return (
                    <td key={levelName} className="px-4 py-3 text-sm
text-gray-700 align-top">
                      {level ? (
                        <>
                          <p>{level.description}</p>
                          {level.points !== undefined && <p
className="font-bold text-gray-800 mt-1">{level.points} pts</p>}
                        </>
                      ) : (
                        <span className="text-gray-400">—</span>
                      )}
                    </td>
                  );
                })}
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
  );
};

// --- Feedback Snippets Management Component ---
/**
 * Allows teachers to manage their reusable feedback snippets and
categories.
 * @param {object} props
 * @param {object} props.onClose
 */
const FeedbackSnippetsManager = ({ onClose }) => {
  const { db, userId, isAuthReady } = useFirebase();
  const [snippets, setSnippets] = useState([]);
  const [newSnippetContent, setNewSnippetContent] = useState('');
  const [newSnippetCategory, setNewSnippetCategory] = useState('');
  const [editingSnippet, setEditingSnippet] = useState(null);
  const [showInfoModal, setShowInfoModal] = useState(false);
  const [infoModalMessage, setInfoModalMessage] = useState('');
  const [generatingCategory, setGeneratingCategory] = useState(false);
  const [showConfirmModal, setShowConfirmModal] = useState(false);
  const [snippetToDelete, setSnippetToDelete] = useState(null);

  const feedbackSnippetsCollection =
`artifacts/${__app_id}/users/${userId}/feedbackSnippets`;

  useEffect(() => {
```

```
    if (!isAuthReady || !db || !userId) return;

    const unsubscribe = onSnapshot(collection(db,
feedbackSnippetsCollection), (snapshot) => {
      setSnippets(snapshot.docs.map(doc => ({ id: doc.id,
...doc.data() })));
    }, (error) => {
      console.error("Error fetching feedback snippets:", error);
      setInfoModalMessage('Failed to load feedback snippets.');
      setShowInfoModal(true);
    });

    return () => unsubscribe();
  }, [db, userId, isAuthReady, feedbackSnippetsCollection]);

  const handleAddSnippet = async () => {
    if (!newSnippetContent.trim()) {
      setInfoModalMessage('Feedback content cannot be empty.');
      setShowInfoModal(true);
      return;
    }
    if (!db || !userId) return;

    try {
      await addDoc(collection(db, feedbackSnippetsCollection), {
        content: newSnippetContent,
        category: newSnippetCategory.trim() || 'General',
        createdAt: new Date(),
      });
      setNewSnippetContent('');
      setNewSnippetCategory('');
      setInfoModalMessage('Feedback snippet added!');
      setShowInfoModal(true);
    } catch (e) {
      console.error("Error adding snippet:", e);
      setInfoModalMessage('Error adding snippet. Try again.');
      setShowInfoModal(true);
    }
  };

  const handleUpdateSnippet = async () => {
    if (!editingSnippet?.content.trim()) {
      setInfoModalMessage('Feedback content cannot be empty.');
      setShowInfoModal(true);
      return;
    }
    if (!db || !userId || !editingSnippet) return;
```

```
    try {
      const snippetRef = doc(db, feedbackSnippetsCollection,
editingSnippet.id);
      await updateDoc(snippetRef, {
        content: editingSnippet.content,
        category: editingSnippet.category.trim() || 'General',
      });
      setEditingSnippet(null);
      setInfoModalMessage('Feedback snippet updated!');
      setShowInfoModal(true);
    } catch (e) {
      console.error("Error updating snippet:", e);
      setInfoModalMessage('Error updating snippet. Try again.');
      setShowInfoModal(true);
    }
  };

  const confirmDeleteSnippet = (snippet) => {
    setSnippetToDelete(snippet);
    setShowConfirmModal(true);
  };

  const handleDeleteSnippet = async () => {
    if (!db || !userId || !snippetToDelete) return;
    try {
      await deleteDoc(doc(db, feedbackSnippetsCollection,
snippetToDelete.id));
      setInfoModalMessage('Feedback snippet deleted!');
      setShowInfoModal(true);
    } catch (e) {
      console.error("Error deleting snippet:", e);
      setInfoModalMessage('Error deleting snippet. Try again.');
      setShowInfoModal(true);
    } finally {
      setShowConfirmModal(false);
      setSnippetToDelete(null);
    }
  };

  const generateCategoryForSnippet = async (contentToCategorize,
isNewSnippet = true) => {
    setGeneratingCategory(true);
    setInfoModalMessage('Analyzing snippet and suggesting
category...');
    setShowInfoModal(true);

    const prompt = `Analyze the following student feedback snippet.
Suggest the single most relevant and concise category for it (e.g.,
```

```
"Grammar", "Thesis Clarity", "Evidence Integration", "Structure",
"Flow", "Formatting", "Argument Development", "Research",
"Engagement", "Effort", "Spelling", "Punctuation"). Provide only the
category name as plain text, no other words or punctuation.

    Feedback Snippet: "${contentToCategorize}"`;

    let chatHistory = [{ role: "user", parts: [{ text: prompt }] }];
    const payload = { contents: chatHistory };
    const apiUrl =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-fl
ash:generateContent?key=${API_KEY}`;

    try {
      const response = await fetch(apiUrl, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      });
      const result = await response.json();

      if (result.candidates && result.candidates.length > 0 &&
          result.candidates[0].content &&
result.candidates[0].content.parts &&
          result.candidates[0].content.parts.length > 0) {
        const generatedCategory =
result.candidates[0].content.parts[0].text.trim();
        if (isNewSnippet) {
          setNewSnippetCategory(generatedCategory);
        } else {
          setEditingSnippet(prev => prev ? { ...prev, category:
generatedCategory } : null);
        }
        setInfoModalMessage('Category suggested successfully!');
      } else {
        setInfoModalMessage('Failed to suggest category. Please try
again.');
      }
    } catch (error) {
      console.error("Error generating category:", error);
      setInfoModalMessage(`Error suggesting category:
${error.message}`);
    } finally {
      setGeneratingCategory(false);
      setShowInfoModal(true);
    }
  };
```

```
  return (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex
items-center justify-center p-4 z-40">
      <div className="bg-white rounded-lg p-6 shadow-xl max-w-2xl
w-full max-h-[90vh] overflow-y-auto">
        <h2 className="text-2xl font-semibold mb-6 text-gray-800
text-center">Manage Feedback Snippets</h2>

        {/* Add New Snippet Form */}
        <div className="mb-8 p-4 border border-blue-200 rounded-lg
bg-blue-50">
          <h3 className="text-xl font-semibold text-blue-800 mb-4">Add
New Snippet</h3>
          <textarea
            value={newSnippetContent}
            onChange={(e) => setNewSnippetContent(e.target.value)}
            placeholder="Enter feedback content..."
            className="w-full p-2 border border-gray-300 rounded-md
mb-2 h-24 resize-y focus:ring-2 focus:ring-blue-500"
          ></textarea>
          <div className="flex items-center space-x-2 mb-4">
            <input
              type="text"
              value={newSnippetCategory}
              onChange={(e) => setNewSnippetCategory(e.target.value)}
              placeholder="Category (e.g., Grammar)"
              className="flex-grow p-2 border border-gray-300
rounded-md focus:ring-2 focus:ring-blue-500"
            />
            <button
              onClick={() =>
generateCategoryForSnippet(newSnippetContent, true)}
              disabled={generatingCategory ||
!newSnippetContent.trim()}
              className={`bg-indigo-500 hover:bg-indigo-600 text-white
font-bold py-2 px-4 rounded-lg transition duration-200
${generatingCategory ? 'opacity-50 cursor-not-allowed' : ''}`}
            >
              {generatingCategory ? 'Analyzing...' : 'Suggest Category
✨'}
            </button>
          </div>
          <button
            onClick={handleAddSnippet}
            className="w-full bg-green-600 hover:bg-green-700
text-white font-bold py-2 px-4 rounded-lg transition duration-200"
          >
            Add Snippet
```

```
          </button>
        </div>

        {/* Snippets List */}
        <div className="mb-8">
          <h3 className="text-xl font-semibold text-gray-800
mb-4">Your Snippets ({snippets.length})</h3>
          {snippets.length === 0 ? (
            <p className="text-gray-600">No snippets added yet.</p>
          ) : (
            <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
              {snippets.map(snippet => (
                <div key={snippet.id} className="bg-gray-50 p-4
rounded-lg border border-gray-200 shadow-sm flex flex-col">
                  {editingSnippet?.id === snippet.id ? (
                    <>
                      <textarea
                        value={editingSnippet.content}
                        onChange={(e) => setEditingSnippet({
...editingSnippet, content: e.target.value })}
                        className="w-full p-2 border border-gray-300
rounded-md mb-2 h-24 resize-y focus:ring-2 focus:ring-blue-500"
                      ></textarea>
                      <div className="flex items-center space-x-2
mb-2">
                        <input
                          type="text"
                          value={editingSnippet.category}
                          onChange={(e) => setEditingSnippet({
...editingSnippet, category: e.target.value })}
                          className="flex-grow p-2 border
border-gray-300 rounded-md focus:ring-2 focus:ring-blue-500"
                        />
                        <button
                          onClick={() =>
generateCategoryForSnippet(editingSnippet.content, false)}
                          disabled={generatingCategory ||
!editingSnippet.content.trim()}
                          className={`bg-indigo-500
hover:bg-indigo-600 text-white font-bold py-1 px-3 rounded-lg text-sm
transition duration-200 ${generatingCategory ? 'opacity-50
cursor-not-allowed' : ''}`}
                        >
                          {generatingCategory ? 'Analyzing...' :
'Suggest ✨'}
                        </button>
                      </div>
                      <div className="flex justify-end space-x-2">
```

```
                        <button
                          onClick={handleUpdateSnippet}
                          className="bg-blue-500 hover:bg-blue-600
text-white text-sm font-bold py-1.5 px-3 rounded-lg"
                        >
                          Save
                        </button>
                        <button
                          onClick={() => setEditingSnippet(null)}
                          className="bg-gray-300 hover:bg-gray-400
text-gray-800 text-sm font-bold py-1.5 px-3 rounded-lg"
                        >
                          Cancel
                        </button>
                      </div>
                    </>
                  ) : (
                    <>
                      <p className="font-semibold text-gray-800
mb-1">{snippet.category}</p>
                      <p className="text-gray-700 text-sm flex-grow
mb-3">{snippet.content}</p>
                      <div className="flex justify-end space-x-2
mt-auto">
                        <button
                          onClick={() => setEditingSnippet({
...snippet })}
                          className="bg-yellow-500 hover:bg-yellow-600
text-white text-xs font-medium py-1.5 px-3 rounded-lg"
                        >
                          Edit
                        </button>
                        <button
                          onClick={() =>
confirmDeleteSnippet(snippet)}
                          className="bg-red-500 hover:bg-red-600
text-white text-xs font-medium py-1.5 px-3 rounded-lg"
                        >
                          Delete
                        </button>
                      </div>
                    </>
                  )}
                </div>
              ))}
            </div>
          )}
        </div>
```

```
        <div className="flex justify-end">
          <button
            onClick={onClose}
            className="bg-gray-500 hover:bg-gray-600 text-white
font-bold py-2 px-5 rounded-lg transition duration-200"
          >
            Close
          </button>
        </div>
      </div>
      {showInfoModal && <InfoModal message={infoModalMessage}
onClose={() => setShowInfoModal(false)} />}
      {showConfirmModal && (
        <ConfirmModal
          message={`Are you sure you want to delete this feedback
snippet?`}
          onConfirm={handleDeleteSnippet}
          onCancel={() => setShowConfirmModal(false)}
        />
      )}
    </div>
  );
};


// --- Assignment Management Components ---

/**
 * Form for adding/editing an assignment. Includes file uploads for
instructions and rubric,
 * and AI-generated rubric capability.
 * @param {object} props
 * @param {string} props.courseId
 * @param {string} props.courseName // Added courseName for AI prompt
 * @param {Function} props.onSave
 * @param {Function} props.onCancel
 * @param {object | null} props.initialAssignmentData
 * @param {object} props.storage Firebase Storage instance.
 */
const AssignmentForm = ({ courseId, courseName, onSave, onCancel,
initialAssignmentData = null, storage }) => {
  const [name, setName] = useState(initialAssignmentData?.name || '');
  const [instructions, setInstructions] =
useState(initialAssignmentData?.instructions || '');
  const [instructionsFile, setInstructionsFile] = useState(null);
  const [instructionsFileUrl, setInstructionsFileUrl] =
useState(initialAssignmentData?.instructionsFileUrl || '');
```

```
  const [rubricFile, setRubricFile] = useState(null);
  const [rubricFileUrl, setRubricFileUrl] =
useState(initialAssignmentData?.rubricFileUrl || '');
  const [rubricData, setRubricData] =
useState(initialAssignmentData?.rubricData ?
JSON.parse(initialAssignmentData.rubricData) : null); // Structured
rubric data
  const [dueDate, setDueDate] =
useState(initialAssignmentData?.dueDate || '');
  const [quizQuestions, setQuizQuestions] =
useState(initialAssignmentData?.quizQuestions ?
JSON.parse(initialAssignmentData.quizQuestions) : []); // New state
for quiz questions
  const [numQuizQuestions, setNumQuizQuestions] = useState(3); //
Default to 3 questions

  const [uploadingInstructions, setUploadingInstructions] =
useState(false);
  const [instructionsUploadProgress, setInstructionsUploadProgress] =
useState(0);
  const [uploadingRubricFile, setUploadingRubricFile] =
useState(false);
  const [rubricFileUploadProgress, setRubricFileUploadProgress] =
useState(0);
  const [generatingRubric, setGeneratingRubric] = useState(false);
  const [generatingIdeas, setGeneratingIdeas] = useState(false);
  const [generatingQuestions, setGeneratingQuestions] =
useState(false); // New state for quiz generation

  const [showInfoModal, setShowInfoModal] = useState(false);
  const [infoModalMessage, setInfoModalMessage] = useState('');

  const handleInstructionsFileChange = (e) => {
    if (e.target.files[0]) {
      setInstructionsFile(e.target.files[0]);
    } else {
      setInstructionsFile(null);
    }
  };

  const handleRubricFileChange = (e) => {
    if (e.target.files[0]) {
      setRubricFile(e.target.files[0]);
      setRubricData(null); // Clear structured data if a file is
uploaded
    } else {
      setRubricFile(null);
    }
```

```
  };

  const generateRubric = async () => {
    setGeneratingRubric(true);
    setInfoModalMessage('Generating sample rubric... This may take a
moment.');
    setShowInfoModal(true);

    const prompt = `Generate a sample academic rubric in JSON format
for an assignment titled "${name || 'General Essay'}". The rubric
should have a title, an optional description, and an array of
criteria. Each criterion should have a name and an array of levels.
Each level should have a 'levelName', 'description', and 'points'.
Include 3-4 criteria with 3-4 levels (e.g., 'Excellent', 'Good',
'Developing', 'Needs Improvement'). Make sure the points decrease for
lower levels. The response should strictly adhere to the JSON schema:
    ${JSON.stringify({
      type: "OBJECT",
      properties: {
        title: { type: "STRING" },
        description: { type: "STRING" },
        criteria: {
          type: "ARRAY",
          items: {
            type: "OBJECT",
            properties: {
              name: { type: "STRING" },
              levels: {
                type: "ARRAY",
                items: {
                  type: "OBJECT",
                  properties: {
                    levelName: { type: "STRING" },
                    description: { type: "STRING" },
                    points: { type: "NUMBER" }
                  }
                }
              }
            }
          }
        }
      },
      propertyOrdering: ["title", "description", "criteria"]
    }, null, 2)}`;

    let chatHistory = [];
    chatHistory.push({ role: "user", parts: [{ text: prompt }] });
    const payload = {
```

```
      contents: chatHistory,
      generationConfig: {
        responseMimeType: "application/json",
        responseSchema: {
          type: "OBJECT",
          properties: {
            "title": { "type": "STRING" },
            "description": { "type": "STRING" },
            "criteria": {
              "type": "ARRAY",
              "items": {
                "type": "OBJECT",
                "properties": {
                  "name": { "type": "STRING" },
                  "levels": {
                    "type": "ARRAY",
                    "items": {
                      "type": "OBJECT",
                      "properties": {
                        "levelName": { "type": "STRING" },
                        "description": { "type": "STRING" },
                        "points": { "type": "NUMBER" }
                      }
                    }
                  }
                }
              }
            },
            "propertyOrdering": ["title", "description", "criteria"]
        }
      }
    };
    const apiUrl =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-fl
ash:generateContent?key=${API_KEY}`;

    try {
      const response = await fetch(apiUrl, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      });

      const result = await response.json(); // Attempt to parse JSON
first

      if (result.candidates && result.candidates.length > 0 &&
```

```
          result.candidates[0].content &&
result.candidates[0].content.parts &&
          result.candidates[0].content.parts.length > 0) {
        const jsonText = result.candidates[0].content.parts[0].text;
        try {
            const parsedRubric = JSON.parse(jsonText);
            setRubricData(parsedRubric);
            setRubricFile(null); // Clear file selection if structured
data is generated
            setRubricFileUrl(''); // Clear file URL if structured data
is generated
            setInfoModalMessage('Rubric generated successfully!');
        } catch (parseError) {
            console.error("Error parsing generated rubric JSON:",
parseError);
            setInfoModalMessage(`Generated rubric is not valid JSON.
Please try again. Error: ${parseError.message}`);
        }
      } else {
        setInfoModalMessage('Failed to generate rubric. Unexpected API
response structure.');
      }
    } catch (error) {
      console.error("Error fetching or processing rubric generation:",
error);
      setInfoModalMessage(`Error generating rubric: ${error.message}.
Please check your network or try again.`);
    } finally {
      setGeneratingRubric(false);
      setShowInfoModal(true);
    }
  };

  const generateAssignmentIdeas = async () => {
    setGeneratingIdeas(true);
    setInfoModalMessage('Generating assignment ideas... This may take
a moment.');
    setShowInfoModal(true);

    const prompt = `Generate 3-5 creative and relevant assignment
ideas for a course titled "${courseName || 'General Course'}" and
topic "${name || 'any topic'}". Each idea should include a brief
description and a suggested type of deliverable (e.g., essay,
presentation, research paper, project). Format as a numbered list.`;

    let chatHistory = [];
    chatHistory.push({ role: "user", parts: [{ text: prompt }] });
    const payload = { contents: chatHistory };
```

```
    const apiUrl =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-fl
ash:generateContent?key=${API_KEY}`;

    try {
      const response = await fetch(apiUrl, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      });
      const result = await response.json();

      if (result.candidates && result.candidates.length > 0 &&
          result.candidates[0].content &&
result.candidates[0].content.parts &&
          result.candidates[0].content.parts.length > 0) {
        const generatedText =
result.candidates[0].content.parts[0].text;
        setInstructions(prev =>
          prev ? `${prev}\n\n--- AI Generated Ideas
---\n${generatedText}` : `--- AI Generated Ideas
---\n${generatedText}`
        );
        setInfoModalMessage('Assignment ideas generated and added to
instructions!');
      } else {
        setInfoModalMessage('Failed to generate assignment ideas.
Please try again.');
      }
    } catch (error) {
      console.error("Error generating assignment ideas:", error);
      setInfoModalMessage(`Error generating assignment ideas:
${error.message}`);
    } finally {
      setGeneratingIdeas(false);
      setShowInfoModal(true);
    }
  };

  const generateQuizQuestions = async () => {
    if (!instructions.trim()) {
      setInfoModalMessage('Please provide assignment instructions
(text) to generate quiz questions.');
      setShowInfoModal(true);
      return;
    }
    if (numQuizQuestions <= 0) {
      setInfoModalMessage('Please enter a valid number of questions
```

```
(greater than 0).');
      setShowInfoModal(true);
      return;
    }

    setGeneratingQuestions(true);
    setInfoModalMessage(`Generating ${numQuizQuestions} quiz
questions... This may take a moment.`);
    setShowInfoModal(true);

    const prompt = `Generate ${numQuizQuestions} multiple-choice quiz
questions based on the following assignment instructions. For each
question, provide the question text, exactly 3 answer options, and
clearly mark one as correct. Ensure the options are plausible
distractors. The response should be in JSON format strictly adhering
to the schema:
    ${JSON.stringify({
      type: "OBJECT",
      properties: {
        quizTitle: { type: "STRING" },
        questions: {
          type: "ARRAY",
          items: {
            type: "OBJECT",
            properties: {
              questionText: { type: "STRING" },
              options: {
                type: "ARRAY",
                items: {
                  type: "OBJECT",
                  properties: {
                    text: { type: "STRING" },
                    isCorrect: { type: "BOOLEAN" }
                  }
                }
              }
            }
          }
        }
      },
      propertyOrdering: ["quizTitle", "questions"]
    }, null, 2)}
    \n\nAssignment Instructions: ${instructions}`;

    let chatHistory = [];
    chatHistory.push({ role: "user", parts: [{ text: prompt }] });
    const payload = {
      contents: chatHistory,
```

```
      generationConfig: {
        responseMimeType: "application/json",
        responseSchema: {
          type: "OBJECT",
          properties: {
            "quizTitle": { "type": "STRING" },
            "questions": {
              "type": "ARRAY",
              "items": {
                "type": "OBJECT",
                "properties": {
                  "questionText": { "type": "STRING" },
                  "options": {
                    "type": "ARRAY",
                    "items": {
                      "type": "OBJECT",
                      "properties": {
                        "text": { "type": "STRING" },
                        "isCorrect": { "type": "BOOLEAN" }
                      }
                    }
                  }
                }
              }
            },
            "propertyOrdering": ["quizTitle", "questions"]
        }
      }
    };
    const apiUrl =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-fl
ash:generateContent?key=${API_KEY}`;

    try {
      const response = await fetch(apiUrl, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      });
      const result = await response.json();

      if (result.candidates && result.candidates.length > 0 &&
          result.candidates[0].content &&
result.candidates[0].content.parts &&
          result.candidates[0].content.parts.length > 0) {
        const jsonText = result.candidates[0].content.parts[0].text;
        try {
```

```
          const parsedQuiz = JSON.parse(jsonText);
          setQuizQuestions(parsedQuiz.questions || []); // Store only
the questions array
          setInfoModalMessage('Quiz questions generated
successfully!');
        } catch (parseError) {
          console.error("Error parsing generated quiz JSON:",
parseError);
          setInfoModalMessage(`Generated quiz is not valid JSON.
Error: ${parseError.message}`);
        }
      } else {
        setInfoModalMessage('Failed to generate quiz questions.
Unexpected API response structure.');
      }
    } catch (error) {
      console.error("Error generating quiz questions:", error);
      setInfoModalMessage(`Error generating quiz questions:
${error.message}. Please check your network or try again.`);
    } finally {
      setGeneratingQuestions(false);
      setShowInfoModal(true);
    }
  };


  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!name.trim() || !dueDate.trim()) {
      setInfoModalMessage('Assignment Name and Due Date are
required.');
      setShowInfoModal(true);
      return;
    }

    let currentInstructionsFileUrl = instructionsFileUrl;
    if (instructionsFile) {
      setUploadingInstructions(true);
      const uploadedURL = await uploadFileToFirebase(
        instructionsFile, storage,
`assignments/${courseId}/instructions/`,
        setInstructionsUploadProgress, setInfoModalMessage,
setShowInfoModal
      );
      if (uploadedURL) {
        currentInstructionsFileUrl = uploadedURL;
      } else {
        setUploadingInstructions(false);
```

```
        return; // Stop submission if upload fails
      }
      setUploadingInstructions(false);
    }

    let currentRubricFileUrl = rubricFileUrl;
    if (rubricFile) {
      setUploadingRubricFile(true);
      const uploadedURL = await uploadFileToFirebase(
        rubricFile, storage, `assignments/${courseId}/rubrics/`,
        setRubricFileUploadProgress, setInfoModalMessage,
setShowInfoModal
      );
      if (uploadedURL) {
        currentRubricFileUrl = uploadedURL;
      } else {
        setUploadingRubricFile(false);
        return; // Stop submission if upload fails
      }
      setUploadingRubricFile(false);
    }

    if (!instructions.trim() && !currentInstructionsFileUrl) {
      setInfoModalMessage('Please provide either instructions text or
upload an instructions file.');
      setShowInfoModal(true);
      return;
    }

    if (!rubricData && !currentRubricFileUrl) {
        setInfoModalMessage('Please provide either a rubric file or
generate/provide structured rubric data.');
        setShowInfoModal(true);
        return;
    }


    onSave({
      name,
      instructions,
      instructionsFileUrl: currentInstructionsFileUrl,
      dueDate,
      rubricFileUrl: currentRubricFileUrl,
      rubricData: rubricData ? JSON.stringify(rubricData) : null, //
Store as string
      quizQuestions: quizQuestions.length > 0 ?
JSON.stringify(quizQuestions) : null, // Store quiz questions
      courseId
```

```
    });
  };

  return (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex
items-center justify-center p-4 z-40">
      <div className="bg-white rounded-lg p-6 shadow-xl max-w-lg
w-full transform transition-all duration-300">
        <h2 className="text-2xl font-semibold mb-6 text-gray-800
text-center">
          {initialAssignmentData ? 'Edit Assignment' : 'Add New
Assignment'}
        </h2>
        <form onSubmit={handleSubmit} className="space-y-4">
          <div>
            <label htmlFor="assignmentName" className="block
text-gray-700 text-sm font-bold mb-2">
              Assignment Name: <span className="text-red-500">*</span>
            </label>
            <input
              type="text"
              id="assignmentName"
              value={name}
              onChange={(e) => setName(e.target.value)}
              className="shadow appearance-none border rounded-lg
w-full py-3 px-4 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition
duration-200"
              placeholder="e.g., Essay: Climate Change Impacts"
            />
          </div>

          <div>
            <label htmlFor="instructions" className="block
text-gray-700 text-sm font-bold mb-2">
              Instructions (Text - Optional):
            </label>
            <textarea
              id="instructions"
              value={instructions}
              onChange={(e) => setInstructions(e.target.value)}
              className="shadow appearance-none border rounded-lg
w-full py-3 px-4 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition
duration-200 h-32 resize-y"
              placeholder="Provide detailed instructions for the
assignment, or upload a file."
            ></textarea>
```

```
            <button
                type="button"
                onClick={generateAssignmentIdeas}
                disabled={generatingIdeas}
                className={`mt-3 w-full font-bold py-2 px-4 rounded-lg
transition duration-200 ease-in-out ${generatingIdeas ? 'bg-indigo-400
cursor-not-allowed' : 'bg-indigo-600 hover:bg-indigo-700
text-white'}`}
            >
                {generatingIdeas ? 'Generating...' : 'Generate
Assignment Ideas (AI) ✨'}
            </button>
            <p className="text-xs text-gray-500 mt-1">
              **Tip:** Found assignment instructions online? Paste
them here and use "Generate Quiz Questions" or "Generate Rubric."
            </p>
          </div>
          <div>
            <label htmlFor="instructionsFile" className="block
text-gray-700 text-sm font-bold mb-2">
              Upload Instructions File (PDF/Doc - Optional):
            </label>
            <input
              type="file"
              id="instructionsFile"
              onChange={handleInstructionsFileChange}
              className="block w-full text-sm text-gray-500
                file:mr-4 file:py-2 file:px-4
                file:rounded-full file:border-0
                file:text-sm file:font-semibold
                file:bg-blue-50 file:text-blue-700
                hover:file:bg-blue-100 cursor-pointer"
            />
            {uploadingInstructions && (
              <div className="w-full bg-gray-200 rounded-full h-2.5
mt-2">
                <div
                  className="bg-blue-600 h-2.5 rounded-full"
                  style={{ width: `${instructionsUploadProgress}%` }}
                ></div>
                <span className="text-xs text-gray-500 mt-1
block">{instructionsUploadProgress.toFixed(0)}% Uploading...</span>
              </div>
            )}
            {instructionsFileUrl && !instructionsFile && (
              <p className="text-sm text-gray-600 mt-2">Current file:
<a href={instructionsFileUrl} target="_blank" rel="noopener
noreferrer" className="text-blue-500 hover:underline break-all">View
```

```
File</a></p>
            )}
          </div>

          <div className="border-t border-gray-200 pt-4 mt-4">
            <label className="block text-gray-700 text-sm font-bold
mb-2">
              Rubric:
            </label>
            <div className="flex space-x-2 mb-4">
              <button
                type="button"
                onClick={generateRubric}
                disabled={generatingRubric}
                className={`flex-1 font-bold py-2 px-4 rounded-lg
transition duration-200 ease-in-out ${generatingRubric ?
'bg-purple-400 cursor-not-allowed' : 'bg-purple-600
hover:bg-purple-700 text-white'}`}
              >
                {generatingRubric ? 'Generating...' : 'Generate Sample
Rubric (AI) ✨'}
              </button>
            </div>
            {rubricData && (
              <div className="mt-4 p-3 bg-gray-50 border
border-gray-200 rounded-lg">
                <h4 className="text-md font-semibold text-gray-800
mb-2">Generated Rubric Preview:</h4>
                <div className="max-h-60 overflow-y-auto">
                    <RubricDisplay rubric={rubricData} />
                </div>
                <button
                    type="button"
                    onClick={() => setRubricData(null)}
                    className="mt-2 text-red-500 hover:text-red-700
text-sm"
                >
                    Clear Generated Rubric
                </button>
              </div>
            )}
            <label htmlFor="rubricFile" className="block text-gray-700
text-sm font-bold mb-2 mt-4">
              Upload Rubric File (PDF/Doc - Optional, overrides
generated):
            </label>
            <input
              type="file"
```

```
              id="rubricFile"
              onChange={handleRubricFileChange}
              className="block w-full text-sm text-gray-500
                file:mr-4 file:py-2 file:px-4
                file:rounded-full file:border-0
                file:text-sm file:font-semibold
                file:bg-blue-50 file:text-blue-700
                hover:file:bg-blue-100 cursor-pointer"
            />
            {uploadingRubricFile && (
              <div className="w-full bg-gray-200 rounded-full h-2.5
mt-2">
                <div
                  className="bg-blue-600 h-2.5 rounded-full"
                  style={{ width: `${rubricFileUploadProgress}%` }}
                ></div>
                <span className="text-xs text-gray-500 mt-1
block">{rubricFileUploadProgress.toFixed(0)}% Uploading...</span>
              </div>
            )}
            {rubricFileUrl && !rubricFile && (
              <p className="text-sm text-gray-600 mt-2">Current file:
<a href={rubricFileUrl} target="_blank" rel="noopener noreferrer"
className="text-blue-500 hover:underline break-all">View File</a></p>
            )}
            <p className="text-xs text-gray-500 mt-1">
              **Tip:** Found a great rubric online? Paste its criteria
into AI rubric generation prompt, or upload it directly here.
            </p>
          </div>

          <div className="border-t border-gray-200 pt-4 mt-4">
            <label className="block text-gray-700 text-sm font-bold
mb-2">
              Quiz Questions:
            </label>
            <div className="flex items-center space-x-2 mb-4">
                <label htmlFor="numQuizQuestions" className="text-sm
text-gray-700">Number of questions:</label>
                <input
                    type="number"
                    id="numQuizQuestions"
                    value={numQuizQuestions}
                    onChange={(e) =>
setNumQuizQuestions(parseInt(e.target.value) || 0)}
                    min="1"
                    max="10" // Limit to 10 for performance/relevance
                    className="w-20 shadow appearance-none border
```

```
rounded-lg py-2 px-3 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 transition duration-200"
                />
              <button
                type="button"
                onClick={generateQuizQuestions}
                disabled={generatingQuestions || !instructions.trim()
|| numQuizQuestions <= 0}
                className={`flex-1 font-bold py-2 px-4 rounded-lg
transition duration-200 ease-in-out ${generatingQuestions ?
'bg-green-400 cursor-not-allowed' : 'bg-green-600 hover:bg-green-700
text-white'}`}
              >
                {generatingQuestions ? 'Generating...' : 'Generate
Quiz Questions (AI) ✨'}
              </button>
            </div>
            {quizQuestions.length > 0 && (
                <div className="mt-4 p-3 bg-gray-50 border
border-gray-200 rounded-lg">
                    <h4 className="text-md font-semibold text-gray-800
mb-2">Generated Quiz Preview:</h4>
                    <div className="max-h-60 overflow-y-auto
space-y-4">
                        {quizQuestions.map((q, qIndex) => (
                            <div key={qIndex} className="text-sm">
                                <p className="font-medium
text-gray-900">{qIndex + 1}. {q.questionText}</p>
                                <ul className="list-disc list-inside
ml-4">
                                    {q.options.map((option, oIndex) =>
(
                                        <li key={oIndex}
className={option.isCorrect ? 'text-green-700 font-semibold' :
'text-gray-700'}>
                                            {option.text}
{option.isCorrect && '(Correct)'}
                                        </li>
                                    ))}
                                </ul>
                            </div>
                        ))}
                    </div>
                    <button
                        type="button"
                        onClick={() => setQuizQuestions([])}
                        className="mt-2 text-red-500
hover:text-red-700 text-sm"
```

```
                    >
                        Clear Generated Quiz
                    </button>
                </div>
            )}
          </div>

          <div>
            <label htmlFor="dueDate" className="block text-gray-700
text-sm font-bold mb-2">
              Due Date: <span className="text-red-500">*</span>
            </label>
            <input
              type="date"
              id="dueDate"
              value={dueDate}
              onChange={(e) => setDueDate(e.target.value)}
              className="shadow appearance-none border rounded-lg
w-full py-3 px-4 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition
duration-200"
            />
          </div>
          <div className="flex justify-end space-x-4 pt-4">
            <button
              type="button"
              onClick={onCancel}
              className="bg-gray-300 hover:bg-gray-400 text-gray-800
font-bold py-2 px-5 rounded-lg transition duration-200 ease-in-out
focus:outline-none focus:ring-2 focus:ring-gray-500
focus:ring-opacity-75"
            >
              Cancel
            </button>
            <button
              type="submit"
              disabled={uploadingInstructions || uploadingRubricFile
|| generatingRubric || generatingIdeas || generatingQuestions}
              className={`font-bold py-2 px-5 rounded-lg transition
duration-200 ease-in-out focus:outline-none focus:ring-2
focus:ring-blue-500 focus:ring-opacity-75 ${uploadingInstructions ||
uploadingRubricFile || generatingRubric || generatingIdeas ||
generatingQuestions ? 'bg-blue-400 cursor-not-allowed' : 'bg-blue-600
hover:bg-blue-700 text-white'}`}
            >
              {initialAssignmentData ? 'Save Changes' : 'Add
Assignment'}
            </button>
```

```
          </div>
        </form>
      </div>
      {showInfoModal && <InfoModal message={infoModalMessage}
onClose={() => setShowInfoModal(false)} />}
    </div>
  );
};

// --- Syllabus Management Components ---

/**
 * Form for adding/editing a syllabus. Includes file upload and AI
summarizer/structurer.
 * @param {object} props
 * @param {string} props.courseId
 * @param {Function} props.onSave
 * @param {Function} props.onCancel
 * @param {object | null} props.initialSyllabusData
 * @param {object} props.storage Firebase Storage instance
 */
const SyllabusForm = ({ courseId, onSave, onCancel,
initialSyllabusData = null, storage }) => {
  const [title, setTitle] = useState(initialSyllabusData?.title ||
'');
  const [content, setContent] = useState(initialSyllabusData?.content
|| '');
  const [file, setFile] = useState(null);
  const [fileUrl, setFileUrl] = useState(initialSyllabusData?.fileUrl
|| '');
  const [uploading, setUploading] = useState(false);
  const [uploadProgress, setUploadProgress] = useState(0);
  const [generatingSummary, setGeneratingSummary] = useState(false);
  const [structuringContent, setStructuringContent] = useState(false);
// New state for structuring

  const [showInfoModal, setShowInfoModal] = useState(false);
  const [infoModalMessage, setInfoModalMessage] = useState('');

  // Handle file selection
  const handleFileChange = (e) => {
    if (e.target.files[0]) {
      setFile(e.target.files[0]);
    } else {
      setFile(null);
    }
  };
```

```
  const generateSyllabusSummary = async () => {
    if (!content.trim()) {
      setInfoModalMessage('Please paste syllabus text content into the
"Content" field to generate a summary. AI cannot read uploaded files
directly.');
      setShowInfoModal(true);
      return;
    }

    setGeneratingSummary(true);
    setInfoModalMessage('Generating syllabus summary... This may take
a moment.');
    setShowInfoModal(true);

    const prompt = `Summarize the following syllabus content,
extracting key policies, important dates, and main learning objectives
concisely. Keep the summary under 300 words.\n\nSyllabus
Content:\n${content}`;

    let chatHistory = [];
    chatHistory.push({ role: "user", parts: [{ text: prompt }] });
    const payload = { contents: chatHistory };
    const apiUrl =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-fl
ash:generateContent?key=${API_KEY}`;

    try {
      const response = await fetch(apiUrl, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      });
      const result = await response.json();

      if (result.candidates && result.candidates.length > 0 &&
          result.candidates[0].content &&
result.candidates[0].content.parts &&
          result.candidates[0].content.parts.length > 0) {
        const generatedText =
result.candidates[0].content.parts[0].text;
        setContent(generatedText); // Replace content with summary
        setInfoModalMessage('Syllabus summary generated
successfully!');
      } else {
        setInfoModalMessage('Failed to generate syllabus summary.
Please try again.');
      }
    } catch (error) {
```

```
      console.error("Error generating syllabus summary:", error);
      setInfoModalMessage(`Error generating syllabus summary:
${error.message}`);
    } finally {
      setGeneratingSummary(false);
      setShowInfoModal(true);
    }
  };

  const structureSyllabusContent = async () => {
    if (!content.trim()) {
      setInfoModalMessage('Please paste syllabus text content into the
"Content" field to structure it. AI cannot read uploaded files
directly.');
      setShowInfoModal(true);
      return;
    }

    setStructuringContent(true);
    setInfoModalMessage('Structuring syllabus content... This may take
a moment.');
    setShowInfoModal(true);

    const prompt = `Analyze the following syllabus content and
reorganize it into standard syllabus sections (e.g., Course
Description, Learning Objectives, Required Materials, Grading Policy,
Schedule, Academic Integrity, Accessibility Statement). If a section
is not present, omit it. Present the output using clear headings
(e.g., "## Course Description").
    \n\nSyllabus Content:\n${content}`;

    let chatHistory = [];
    chatHistory.push({ role: "user", parts: [{ text: prompt }] });
    const payload = { contents: chatHistory };
    const apiUrl =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-fl
ash:generateContent?key=${API_KEY}`;

    try {
      const response = await fetch(apiUrl, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      });
      const result = await response.json();

      if (result.candidates && result.candidates.length > 0 &&
          result.candidates[0].content &&
```

```
result.candidates[0].content.parts &&
          result.candidates[0].content.parts.length > 0) {
        const generatedText =
result.candidates[0].content.parts[0].text;
        setContent(generatedText); // Replace content with structured
text
        setInfoModalMessage('Syllabus content structured
successfully!');
      } else {
        setInfoModalMessage('Failed to structure syllabus content.
Please try again.');
      }
    } catch (error) {
      console.error("Error structuring syllabus content:", error);
      setInfoModalMessage(`Error structuring syllabus content:
${error.message}`);
    } finally {
      setStructuringContent(false);
      setShowInfoModal(true);
    }
  };


  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!title.trim()) {
      setInfoModalMessage('Syllabus title is required.');
      setShowInfoModal(true);
      return;
    }

    let currentFileUrl = fileUrl;
    if (file) {
      setUploading(true);
      const uploadedURL = await uploadFileToFirebase(
        file, storage, `syllabi/${courseId}/`,
        setUploadProgress, setInfoModalMessage, setShowInfoModal
      );
      if (uploadedURL) {
        currentFileUrl = uploadedURL;
      } else {
        setUploading(false);
        return;
      }
      setUploading(false);
    } else if (!content.trim() && !currentFileUrl) {
      setInfoModalMessage('Please provide either text content or
upload a file for the syllabus.');
```

```
      setShowInfoModal(true);
      return;
    }

    onSave({ title, content, fileUrl: currentFileUrl, courseId });
  };

  return (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex
items-center justify-center p-4 z-40">
      <div className="bg-white rounded-lg p-6 shadow-xl max-w-lg
w-full transform transition-all duration-300">
        <h2 className="text-2xl font-semibold mb-6 text-gray-800
text-center">
          {initialSyllabusData ? 'Edit Syllabus' : 'Add New Syllabus'}
        </h2>
        <form onSubmit={handleSubmit} className="space-y-4">
          <div>
            <label htmlFor="syllabusTitle" className="block
text-gray-700 text-sm font-bold mb-2">
              Title: <span className="text-red-500">*</span>
            </label>
            <input
              type="text"
              id="syllabusTitle"
              value={title}
              onChange={(e) => setTitle(e.target.value)}
              className="shadow appearance-none border rounded-lg
w-full py-3 px-4 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition
duration-200"
              placeholder="e.g., Fall 2024 Syllabus"
            />
          </div>
          <div>
            <label htmlFor="syllabusContent" className="block
text-gray-700 text-sm font-bold mb-2">
              Content (Text - Optional):
            </label>
            <textarea
              id="syllabusContent"
              value={content}
              onChange={(e) => setContent(e.target.value)}
              className="shadow appearance-none border rounded-lg
w-full py-3 px-4 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition
duration-200 h-48 resize-y"
              placeholder="Paste syllabus content here (e.g., course
```

```
policies, grading rubric, schedule) OR upload a file."
            ></textarea>
            <div className="flex space-x-2 mt-3">
                <button
                    type="button"
                    onClick={generateSyllabusSummary}
                    disabled={generatingSummary || !content.trim() ||
structuringContent}
                    className={`flex-1 font-bold py-2 px-4 rounded-lg
transition duration-200 ease-in-out ${generatingSummary ||
structuringContent ? 'bg-indigo-400 cursor-not-allowed' :
'bg-indigo-600 hover:bg-indigo-700 text-white'}`}
                >
                    {generatingSummary ? 'Summarizing...' : 'Summarize
Syllabus (AI) ✨'}
                </button>
                <button
                    type="button"
                    onClick={structureSyllabusContent}
                    disabled={structuringContent || !content.trim() ||
generatingSummary}
                    className={`flex-1 font-bold py-2 px-4 rounded-lg
transition duration-200 ease-in-out ${structuringContent ||
generatingSummary ? 'bg-indigo-400 cursor-not-allowed' :
'bg-indigo-600 hover:bg-indigo-700 text-white'}`}
                >
                    {structuringContent ? 'Structuring...' :
'Structure Syllabus (AI) ✨'}
                </button>
            </div>
            <p className="text-xs text-gray-500 mt-1">
              **Tip:** Found an open-source syllabus? Paste its
content here, then use AI to summarize or structure it!
            </p>
          </div>
          <div>
            <label htmlFor="syllabusFile" className="block
text-gray-700 text-sm font-bold mb-2">
              Upload Syllabus File (PDF/Doc - Optional):
            </label>
            <input
              type="file"
              id="syllabusFile"
              onChange={handleFileChange}
              className="block w-full text-sm text-gray-500
                file:mr-4 file:py-2 file:px-4
                file:rounded-full file:border-0
                file:text-sm file:font-semibold
```

```
                file:bg-blue-50 file:text-blue-700
                hover:file:bg-blue-100 cursor-pointer"
              />
              {uploading && (
                <div className="w-full bg-gray-200 rounded-full h-2.5
mt-2">
                  <div
                    className="bg-blue-600 h-2.5 rounded-full"
                    style={{ width: `${uploadProgress}%` }}
                  ></div>
                  <span className="text-xs text-gray-500 mt-1
block">{uploadProgress.toFixed(0)}% Uploading...</span>
                </div>
              )}
              {fileUrl && !file && (
                <p className="text-sm text-gray-600 mt-2">Current file:
<a href={fileUrl} target="_blank" rel="noopener noreferrer"
className="text-blue-500 hover:underline break-all">View File</a></p>
              )}
            </div>
            <div className="flex justify-end space-x-4 pt-4">
              <button
                type="button"
                onClick={onCancel}
                className="bg-gray-300 hover:bg-gray-400 text-gray-800
font-bold py-2 px-5 rounded-lg transition duration-200 ease-in-out
focus:outline-none focus:ring-2 focus:ring-gray-500
focus:ring-opacity-75"
              >
                Cancel
              </button>
              <button
                type="submit"
                disabled={uploading || generatingSummary ||
structuringContent}
                className={`font-bold py-2 px-5 rounded-lg transition
duration-200 ease-in-out focus:outline-none focus:ring-2
focus:ring-blue-500 focus:ring-opacity-75 ${uploading ||
generatingSummary || structuringContent ? 'bg-blue-400
cursor-not-allowed' : 'bg-blue-600 hover:bg-blue-700 text-white'}`}
              >
                {initialSyllabusData ? 'Save Changes' : 'Add Syllabus'}
              </button>
            </div>
          </form>
        </div>
        {showInfoModal && <InfoModal message={infoModalMessage}
onClose={() => setShowInfoModal(false)} />}
```

```
    </div>
  );
};

// --- Student Sample Management Components ---

/**
 * Form for adding/editing a student sample, feedback, and grade.
Includes file upload and AI feedback.
 * @param {object} props
 * @param {string} props.courseId
 * @param {Array<object>} props.allAssignments List of all assignments
for the current course.
 * @param {Function} props.onSave
 * @param {Function} props.onCancel
 * @param {object | null} props.initialSampleData
 * @param {object} props.storage Firebase Storage instance.
 * @param {Array<object>} props.feedbackSnippets Global feedback
snippets for reuse.
 */
const StudentSampleForm = ({ courseId, allAssignments, onSave,
onCancel, initialSampleData = null, storage, feedbackSnippets }) => {
  const [studentName, setStudentName] =
useState(initialSampleData?.studentName || '');
  const [sampleContent, setSampleContent] =
useState(initialSampleData?.sampleContent || '');
  const [sampleFile, setSampleFile] = useState(null);
  const [sampleFileUrl, setSampleFileUrl] =
useState(initialSampleData?.sampleFileUrl || '');
  const [feedback, setFeedback] = useState(initialSampleData?.feedback
|| '');
  const [grade, setGrade] = useState(initialSampleData?.grade || '');
  const [selectedAssignmentId, setSelectedAssignmentId] =
useState(initialSampleData?.assignmentId || '');
  const [selectedAssignment, setSelectedAssignment] = useState(null);
// Full assignment object

  const [uploadingSample, setUploadingSample] = useState(false);
  const [sampleUploadProgress, setSampleUploadProgress] = useState(0);
  const [generatingFeedback, setGeneratingFeedback] = useState(false);
  const [refiningFeedback, setRefiningFeedback] = useState(false); //
New state for feedback refinement

  const [showInfoModal, setShowInfoModal] = useState(false);
  const [infoModalMessage, setInfoModalMessage] = useState('');
  const [showFeedbackSnippetBrowser, setShowFeedbackSnippetBrowser] =
useState(false); // State to show snippet browser
```

```
  // Effect to set the selected assignment object when
selectedAssignmentId changes
  useEffect(() => {
    if (selectedAssignmentId && allAssignments.length > 0) {
      const assignment = allAssignments.find(a => a.id ===
selectedAssignmentId);
      setSelectedAssignment(assignment);
    } else {
      setSelectedAssignment(null);
    }
  }, [selectedAssignmentId, allAssignments]);


  const handleSampleFileChange = (e) => {
    if (e.target.files[0]) {
      setSampleFile(e.target.files[0]);
    } else {
      setSampleFile(null);
    }
  };

  const generateStudentFeedback = async () => {
    if (!selectedAssignment) {
      setInfoModalMessage('Please select an assignment to generate
feedback.');
      setShowInfoModal(true);
      return;
    }
    if (!sampleContent.trim()) {
      setInfoModalMessage('Please provide student sample content
(text) to generate feedback. AI cannot read uploaded files yet.');
      setShowInfoModal(true);
      return;
    }
    if (!grade.trim()) {
        setInfoModalMessage('Please provide a grade to generate
feedback.');
        setShowInfoModal(true);
        return;
    }

    setGeneratingFeedback(true);
    setInfoModalMessage('Generating feedback... This may take a
moment.');
    setShowInfoModal(true);

    let prompt = `As a teacher, provide constructive feedback for a
student named "${studentName || 'the student'}" for their assignment
```

```
titled "${selectedAssignment.name}".\n\n`;
    prompt += `Student's work: "${sampleContent}"\n\n`;
    prompt += `Assigned Grade: ${grade}\n\n`;

    if (selectedAssignment.instructions) {
      prompt += `Assignment Instructions:
"${selectedAssignment.instructions}"\n\n`;
    }
    if (selectedAssignment.rubricData) {
      try {
        const parsedRubric =
JSON.parse(selectedAssignment.rubricData);
        prompt += `Rubric:\n${JSON.stringify(parsedRubric, null,
2)}\n\n`;
      } catch (e) {
        console.error("Error parsing rubric data for feedback
generation:", e);
      }
    }

    if (feedbackSnippets.length > 0) {
      prompt += `Consider the following common feedback themes you use
(categorized): ${JSON.stringify(feedbackSnippets.map(s => ({ category:
s.category, content: s.content })))}\n\n`;
      prompt += `Please provide feedback that is encouraging but also
identifies areas for improvement based on the work, instructions, and
rubric (if provided). Incorporate relevant themes from the provided
common feedback examples. Keep the tone appropriate for a student.`;
    } else {
      prompt += `Please provide feedback that is encouraging but also
identifies areas for improvement based on the work, instructions, and
rubric (if provided). Keep the tone appropriate for a student.`;
    }


    let chatHistory = [];
    chatHistory.push({ role: "user", parts: [{ text: prompt }] });
    const payload = {
      contents: chatHistory,
      // No responseSchema needed for free-form text feedback
    };
    const apiUrl =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-fl
ash:generateContent?key=${API_KEY}`;

    try {
      const response = await fetch(apiUrl, {
        method: 'POST',
```

```
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      });
      const result = await response.json();

      if (result.candidates && result.candidates.length > 0 &&
          result.candidates[0].content &&
result.candidates[0].content.parts &&
          result.candidates[0].content.parts.length > 0) {
        const generatedText =
result.candidates[0].content.parts[0].text;
        setFeedback(generatedText);
        setInfoModalMessage('Feedback generated successfully!');
      } else {
        setInfoModalMessage('Failed to generate feedback. Please try
again.');
      }
    } catch (error) {
      console.error("Error generating feedback:", error);
      setInfoModalMessage(`Error generating feedback:
${error.message}`);
    } finally {
      setGeneratingFeedback(false);
      setShowInfoModal(true);
    }
  };

  const refineFeedback = async () => {
    if (!feedback.trim()) {
      setInfoModalMessage('There is no feedback to refine. Generate or
type some feedback first.');
      setShowInfoModal(true);
      return;
    }
    setRefiningFeedback(true);
    setInfoModalMessage('Refining feedback...');
    setShowInfoModal(true);

    const prompt = `Refine the following student feedback. You can
adjust the tone (more encouraging, more direct), expand on points, or
make it more specific. Keep it professional and constructive. Current
Feedback: "${feedback}"`;

    let chatHistory = [];
    chatHistory.push({ role: "user", parts: [{ text: prompt }] });
    const payload = { contents: chatHistory };
    const apiUrl =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-fl
```

```
ash:generateContent?key=${API_KEY}`;

    try {
      const response = await fetch(apiUrl, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      });
      const result = await response.json();

      if (result.candidates && result.candidates.length > 0 &&
          result.candidates[0].content &&
result.candidates[0].content.parts &&
          result.candidates[0].content.parts.length > 0) {
        const refinedText =
result.candidates[0].content.parts[0].text;
        setFeedback(refinedText);
        setInfoModalMessage('Feedback refined successfully!');
      } else {
        setInfoModalMessage('Failed to refine feedback. Please try
again.');
      }
    } catch (error) {
      console.error("Error refining feedback:", error);
      setInfoModalMessage(`Error refining feedback:
${error.message}`);
    } finally {
      setRefiningFeedback(false);
      setShowInfoModal(true);
    }
  };

  const insertFeedbackSnippet = (snippetContent) => {
    setFeedback(prev => prev ? `${prev}\n\n${snippetContent}` :
snippetContent);
    setShowFeedbackSnippetBrowser(false); // Close browser after
insertion
    setInfoModalMessage('Snippet inserted into feedback.');
    setShowInfoModal(true);
  };


  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!studentName.trim() || !grade.trim() || !selectedAssignmentId)
{
      setInfoModalMessage('Student Name, Grade, and Assignment are
required.');
```

```
      setShowInfoModal(true);
      return;
    }

    let currentSampleFileUrl = sampleFileUrl;
    if (sampleFile) {
      setUploadingSample(true);
      const uploadedURL = await uploadFileToFirebase(
        sampleFile, storage,
`studentSamples/${courseId}/${selectedAssignmentId}/`,
        setSampleUploadProgress, setInfoModalMessage, setShowInfoModal
      );
      if (uploadedURL) {
        currentSampleFileUrl = uploadedURL;
      } else {
        setUploadingSample(false);
        return;
      }
      setUploadingSample(false);
    } else if (!sampleContent.trim() && !currentSampleFileUrl) {
      setInfoModalMessage('Please provide either sample text content
or upload a sample file.');
      setShowInfoModal(true);
      return;
    }

    onSave({
      studentName,
      sampleContent,
      sampleFileUrl: currentSampleFileUrl,
      feedback,
      grade,
      courseId,
      assignmentId: selectedAssignmentId
    });
  };

  return (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex
items-center justify-center p-4 z-40">
      <div className="bg-white rounded-lg p-6 shadow-xl max-w_lg
w-full transform transition-all duration-300">
        <h2 className="text-2xl font-semibold mb-6 text-gray-800
text-center">
          {initialSampleData ? 'Edit Student Sample' : 'Add New
Student Sample'}
        </h2>
        <form onSubmit={handleSubmit} className="space-y-4">
```

```
          <div>
            <label htmlFor="studentName" className="block
text-gray-700 text-sm font-bold mb-2">
              Student Name: <span className="text-red-500">*</span>
            </label>
            <input
              type="text"
              id="studentName"
              value={studentName}
              onChange={(e) => setStudentName(e.target.value)}
              className="shadow appearance-none border rounded-lg
w-full py-3 px-4 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition
duration-200"
              placeholder="e.g., Alice Smith"
            />
          </div>

          <div>
            <label htmlFor="assignmentSelect" className="block
text-gray-700 text-sm font-bold mb-2">
              Link to Assignment: <span
className="text-red-500">*</span>
            </label>
            <select
              id="assignmentSelect"
              value={selectedAssignmentId}
              onChange={(e) =>
setSelectedAssignmentId(e.target.value)}
              className="shadow appearance-none border rounded-lg
w-full py-3 px-4 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition
duration-200"
            >
              <option value="">-- Select an Assignment --</option>
              {allAssignments.map(assignment => (
                <option key={assignment.id} value={assignment.id}>
                  {assignment.name} (Due: {assignment.dueDate})
                </option>
              ))}
            </select>
            {allAssignments.length === 0 && (
              <p className="text-sm text-red-500 mt-1">No assignments
found. Please add an assignment first.</p>
            )}
          </div>

          <div>
```

```
            <label htmlFor="sampleContent" className="block
text-gray-700 text-sm font-bold mb-2">
              Student Sample Content (Text - Optional, recommended for
AI feedback):
            </label>
            <textarea
              id="sampleContent"
              value={sampleContent}
              onChange={(e) => setSampleContent(e.target.value)}
              className="shadow appearance-none border rounded-lg
w-full py-3 px-4 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition
duration-200 h-32 resize-y"
              placeholder="Paste student's work here, or upload a
file. Text content is required for AI feedback generation."
            ></textarea>
          </div>
          <div>
            <label htmlFor="sampleFile" className="block text-gray-700
text-sm font-bold mb-2">
              Upload Student Sample File (PDF/Doc - Optional):
            </label>
            <input
              type="file"
              id="sampleFile"
              onChange={handleSampleFileChange}
              className="block w-full text-sm text-gray-500
                file:mr-4 file:py-2 file:px-4
                file:rounded-full file:border-0
                file:text-sm file:font-semibold
                file:bg-blue-50 file:text-blue-700
                hover:file:bg-blue-100 cursor-pointer"
            />
            {uploadingSample && (
              <div className="w-full bg-gray-200 rounded-full h-2.5
mt-2">
                <div
                  className="bg-blue-600 h-2.5 rounded-full"
                  style={{ width: `${sampleUploadProgress}%` }}
                ></div>
                <span className="text-xs text-gray-500 mt-1
block">{sampleUploadProgress.toFixed(0)}% Uploading...</span>
              </div>
            )}
            {sampleFileUrl && !sampleFile && (
              <p className="text-sm text-gray-600 mt-2">Current file:
<a href={sampleFileUrl} target="_blank" rel="noopener noreferrer"
className="text-blue-500 hover:underline break-all">View File</a></p>
```

```
            )}
          </div>
          <div>
            <label htmlFor="grade" className="block text-gray-700
text-sm font-bold mb-2">
              Grade: <span className="text-red-500">*</span>
            </label>
            <input
              type="text"
              id="grade"
              value={grade}
              onChange={(e) => setGrade(e.target.value)}
              className="shadow appearance-none border rounded-lg
w-full py-3 px-4 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition
duration-200"
              placeholder="e.g., A+, 95/100, Excellent"
            />
          </div>
          <div>
            <label htmlFor="feedback" className="block text-gray-700
text-sm font-bold mb-2">
              Feedback (Optional):
            </label>
            <textarea
              id="feedback"
              value={feedback}
              onChange={(e) => setFeedback(e.target.value)}
              className="shadow appearance-none border rounded-lg
w-full py-3 px-4 text-gray-700 leading-tight focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition
duration-200 h-24 resize-y"
              placeholder="Provide detailed feedback for the student,
or generate using AI."
            ></textarea>
            <div className="flex space-x-2 mt-3">
                <button
                type="button"
                onClick={generateStudentFeedback}
                disabled={generatingFeedback || !selectedAssignment ||
!sampleContent.trim() || !grade.trim()}
                className={`flex-1 font-bold py-2 px-4 rounded-lg
transition duration-200 ease-in-out ${generatingFeedback ?
'bg-purple-400 cursor-not-allowed' : 'bg-purple-600
hover:bg-purple-700 text-white'}`}
                >
                {generatingFeedback ? 'Generating...' : 'Generate
Feedback (AI) ✨'}
```

```
                </button>
                <button
                type="button"
                onClick={refineFeedback}
                disabled={refiningFeedback || !feedback.trim()}
                className={`flex-1 font-bold py-2 px-4 rounded-lg
transition duration-200 ease-in-out ${refiningFeedback ? 'bg-teal-400
cursor-not-allowed' : 'bg-teal-600 hover:bg-teal-700 text-white'}`}
                >
                {refiningFeedback ? 'Refining...' : 'Refine Feedback
(AI) ✨'}
                </button>
            </div>
            <button
                type="button"
                onClick={() => setShowFeedbackSnippetBrowser(true)}
                className="mt-3 w-full bg-blue-500 hover:bg-blue-600
text-white font-bold py-2 px-4 rounded-lg transition duration-200"
            >
                Browse Feedback Snippets
            </button>
            <p className="text-xs text-gray-500 mt-1">
              Note: AI feedback generation requires student sample
content (text), assignment selection, and a grade.
            </p>
          </div>
          <div className="flex justify-end space-x-4 pt-4">
            <button
              type="button"
              onClick={onCancel}
              className="bg-gray-300 hover:bg-gray-400 text-gray-800
font-bold py-2 px-5 rounded-lg transition duration-200 ease-in-out
focus:outline-none focus:ring-2 focus:ring-gray-500
focus:ring-opacity-75"
            >
              Cancel
            </button>
            <button
              type="submit"
              disabled={uploadingSample || generatingFeedback ||
refiningFeedback}
              className={`font-bold py-2 px-5 rounded-lg transition
duration-200 ease-in-out focus:outline-none focus:ring-2
focus:ring-blue-500 focus:ring-opacity-75 ${uploadingSample ||
generatingFeedback || refiningFeedback ? 'bg-blue-400
cursor-not-allowed' : 'bg-blue-600 hover:bg-blue-700 text-white'}`}
            >
              {initialSampleData ? 'Save Changes' : 'Add Sample'}
```

```
            </button>
          </div>
        </form>
      </div>
      {showInfoModal && <InfoModal message={infoModalMessage}
onClose={() => setShowInfoModal(false)} />}
      {showFeedbackSnippetBrowser && (
        <FeedbackSnippetBrowser
          feedbackSnippets={feedbackSnippets}
          onSelectSnippet={insertFeedbackSnippet}
          onClose={() => setShowFeedbackSnippetBrowser(false)}
        />
      )}
    </div>
  );
};

// --- Feedback Snippet Browser Component ---
/**
 * Allows browsing and selecting feedback snippets from the library.
 * @param {object} props
 * @param {Array<object>} props.feedbackSnippets
 * @param {Function} props.onSelectSnippet
 * @param {Function} props.onClose
 */
const FeedbackSnippetBrowser = ({ feedbackSnippets, onSelectSnippet,
onClose }) => {
    const [searchTerm, setSearchTerm] = useState('');
    const [selectedCategory, setSelectedCategory] = useState('All');

    const categories = ['All', ...new Set(feedbackSnippets.map(s =>
s.category))].sort(); // Sort categories alphabetically

    const filteredSnippets = feedbackSnippets.filter(snippet => {
        const matchesSearch =
snippet.content.toLowerCase().includes(searchTerm.toLowerCase()) ||

snippet.category.toLowerCase().includes(searchTerm.toLowerCase());
        const matchesCategory = selectedCategory === 'All' ||
snippet.category === selectedCategory;
        return matchesSearch && matchesCategory;
    }).sort((a, b) => a.category.localeCompare(b.category) ||
a.content.localeCompare(b.content)); // Sort by category then content

    return (
        <div className="fixed inset-0 bg-black bg-opacity-70 flex
items-center justify-center p-4 z-50">
            <div className="bg-white rounded-lg p-6 shadow-xl
```

```
max-w-3xl w-full max-h-[90vh] overflow-y-auto">
                <h2 className="text-2xl font-semibold mb-6
text-gray-800 text-center">Browse Feedback Snippets</h2>

                <div className="flex flex-col sm:flex-row space-y-3
sm:space-y-0 sm:space-x-4 mb-6">
                    <input
                        type="text"
                        placeholder="Search snippets..."
                        value={searchTerm}
                        onChange={(e) =>
setSearchTerm(e.target.value)}
                        className="flex-grow p-2 border
border-gray-300 rounded-md focus:ring-2 focus:ring-blue-500"
                    />
                    <select
                        value={selectedCategory}
                        onChange={(e) =>
setSelectedCategory(e.target.value)}
                        className="p-2 border border-gray-300
rounded-md focus:ring-2 focus:ring-blue-500"
                    >
                        {categories.map(cat => (
                            <option key={cat}
value={cat}>{cat}</option>
                        ))}
                    </select>
                </div>

                {filteredSnippets.length === 0 ? (
                    <p className="text-center text-gray-600">No
snippets found matching your criteria.</p>
                ) : (
                    <div className="grid grid-cols-1 md:grid-cols-2
gap-4 max-h-80 overflow-y-auto mb-6">
                        {filteredSnippets.map(snippet => (
                            <div key={snippet.id}
                                 className="bg-gray-50 p-4 rounded-lg
border border-gray-200 shadow-sm cursor-pointer hover:bg-blue-50
hover:border-blue-300 transition"
                                 onClick={() =>
onSelectSnippet(snippet.content)}>
                                <p className="font-semibold
text-blue-800 text-sm mb-1">{snippet.category}</p>
                                <p className="text-gray-800 text-sm
line-clamp-3">{snippet.content}</p>
                            </div>
                        ))}
```

```
                    </div>
                )}

                <div className="flex justify-end">
                    <button
                        onClick={onClose}
                        className="bg-gray-500 hover:bg-gray-600
text-white font-bold py-2 px-5 rounded-lg transition duration-200"
                    >
                        Close
                    </button>
                </div>
            </div>
        </div>
    );
};

// --- Feedback Insights Dashboard ---
const FeedbackInsights = ({ onClose, allStudentSamples, allAssignments
}) => {
  const { db, userId, isAuthReady } = useFirebase();
  const [insights, setInsights] = useState(null);
  const [generatingInsights, setGeneratingInsights] = useState(false);
  const [showInfoModal, setShowInfoModal] = useState(false);
  const [infoModalMessage, setInfoModalMessage] = useState('');

  const generateInsights = async () => {
    if (allStudentSamples.length === 0) {
      setInfoModalMessage('No student samples with feedback found to
generate insights.');
      setShowInfoModal(true);
      return;
    }

    setGeneratingInsights(true);
    setInsights(null);
    setInfoModalMessage('Generating feedback insights... This may take
a moment.');
    setShowInfoModal(true);

    const feedbackDataForAI = allStudentSamples.map(sample => {
      const assignmentName = allAssignments.find(a => a.id ===
sample.assignmentId)?.name || 'Unknown Assignment';
      return {
        studentName: sample.studentName,
        assignment: assignmentName,
        grade: sample.grade,
        feedback: sample.feedback
```

```
      };
    }).filter(item => item.feedback && item.feedback.trim() !== '');
// Filter out empty feedback

    if (feedbackDataForAI.length === 0) {
      setInfoModalMessage('No feedback available across student
samples to generate insights.');
      setShowInfoModal(true);
      setGeneratingInsights(false);
      return;
    }


    const prompt = `Analyze the following collection of student
feedback entries. Identify the top 5 most common areas of strength and
top 5 most common areas for improvement observed across these
submissions. Focus on academic writing aspects like thesis
development, evidence use, organization, clarity, and grammar. Also
identify any general trends you notice across assignments or students.
Present the strengths and weaknesses as two distinct numbered lists
with brief explanations. Then provide a paragraph summarizing general
trends.

    Feedback Data: ${JSON.stringify(feedbackDataForAI, null, 2)}`;

    let chatHistory = [{ role: "user", parts: [{ text: prompt }] }];
    const payload = { contents: chatHistory };
    const apiUrl =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-fl
ash:generateContent?key=${API_KEY}`;

    try {
      const response = await fetch(apiUrl, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      });
      const result = await response.json();

      if (result.candidates && result.candidates.length > 0 &&
          result.candidates[0].content &&
result.candidates[0].content.parts &&
          result.candidates[0].content.parts.length > 0) { //
Corrected access to generated content
        const generatedText =
result.candidates[0].content.parts[0].text;
        setInsights(generatedText);
        setInfoModalMessage('Feedback insights generated!');
```

```
      } else {
        setInfoModalMessage('Failed to generate insights. Unexpected
API response structure.');
      }
    } catch (error) {
      console.error("Error generating insights:", error);
      setInfoModalMessage(`Error generating insights:
${error.message}. Please check your network or try again.`);
    } finally {
      setGeneratingInsights(false);
      setShowInfoModal(true);
    }
  };

  return (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex
items-center justify-center p-4 z-50">
      <div className="bg-white rounded-lg p-6 shadow-xl max-w-3xl
w-full max-h-[90vh] overflow-y-auto">
        <h2 className="text-2xl font-semibold mb-6 text-gray-800
text-center">Feedback Insights</h2>

        <div className="flex justify-center mb-6">
          <button
            onClick={generateInsights}
            disabled={generatingInsights || allStudentSamples.length
=== 0}
            className={`bg-purple-600 hover:bg-purple-700 text-white
font-bold py-2 px-5 rounded-lg transition duration-200
${generatingInsights ? 'opacity-50 cursor-not-allowed' : ''}`}
          >
            {generatingInsights ? 'Analyzing...' : 'Generate Overall
Insights ✨'}
          </button>
        </div>

        {insights ? (
          <div className="bg-gray-50 p-4 rounded-lg border
border-gray-200 whitespace-pre-wrap">
            {insights}
          </div>
        ) : (
          <p className="text-center text-gray-600">Click "Generate
Overall Insights" to analyze feedback across all student samples.</p>
        )}

        <div className="flex justify-end mt-6">
          <button
```

```
            onClick={onClose}
            className="bg-gray-500 hover:bg-gray-600 text-white
font-bold py-2 px-5 rounded-lg transition duration-200"
          >
            Close
          </button>
        </div>
      </div>
      {showInfoModal && <InfoModal message={infoModalMessage}
onClose={() => setShowInfoModal(false)} />}
    </div>
  );
};


// --- Course Detail Component (combines assignments, syllabi,
samples) ---

/**
 * Displays details of a specific course, including assignments,
syllabi, and student samples.
 * Allows adding/editing/deleting these items.
 * @param {object} props
 * @param {object} props.course
 * @param {Function} props.onBack
 */
const CourseDetail = ({ course, onBack }) => {
  const { db, userId, isAuthReady, storage } = useFirebase();
  const [assignments, setAssignments] = useState([]);
  const [syllabi, setSyllabi] = useState([]);
  const [studentSamples, setStudentSamples] = useState([]);
  const [feedbackSnippets, setFeedbackSnippets] = useState([]); //
State for global feedback snippets

  const [showAddAssignmentForm, setShowAddAssignmentForm] =
useState(false);
  const [editingAssignment, setEditingAssignment] = useState(null);
  const [showAddSyllabusForm, setShowAddSyllabusForm] =
useState(false);
  const [editingSyllabus, setEditingSyllabus] = useState(null);
  const [showAddSampleForm, setShowAddSampleForm] = useState(false);
  const [editingSample, setEditingSample] = useState(null);
  const [showConfirmModal, setShowConfirmModal] = useState(false);
  const [itemToDelete, setItemToDelete] = useState(null);
  const [deleteType, setDeleteType] = useState(''); // 'assignment',
'syllabus', 'sample'
  const [showInfoModal, setShowInfoModal] = useState(false);
  const [infoModalMessage, setInfoModalMessage] = useState('');
```

```
  const [showFeedbackManager, setShowFeedbackManager] =
useState(false); // State for feedback manager modal
  const [showFeedbackInsights, setShowFeedbackInsights] =
useState(false); // State for feedback insights modal


  const collections = {
    assignments: `artifacts/${__app_id}/users/${userId}/assignments`,
    syllabi: `artifacts/${__app_id}/users/${userId}/syllabi`,
    studentSamples:
`artifacts/${__app_id}/users/${userId}/studentSamples`,
    feedbackSnippets:
`artifacts/${__app_id}/users/${userId}/feedbackSnippets`, // New
collection
  };

  const fetchData = useCallback(() => {
    if (!isAuthReady || !db || !userId || !course?.id) {
      return;
    }

    // Fetch Assignments
    const assignmentsQuery = query(collection(db,
collections.assignments), where("courseId", "==", course.id));
    onSnapshot(assignmentsQuery, (snapshot) => {
      setAssignments(snapshot.docs.map(doc => ({ id: doc.id,
...doc.data() })));
    }, (error) => {
      console.error("Error fetching assignments:", error);
      setInfoModalMessage('Failed to load assignments.');
      setShowInfoModal(true);
    });

    // Fetch Syllabi
    const syllabiQuery = query(collection(db, collections.syllabi),
where("courseId", "==", course.id));
    onSnapshot(syllabiQuery, (snapshot) => {
      setSyllabi(snapshot.docs.map(doc => ({ id: doc.id, ...doc.data()
})));
    }, (error) => {
      console.error("Error fetching syllabi:", error);
      setInfoModalMessage('Failed to load syllabi.');
      setShowInfoModal(true);
    });

    // Fetch Student Samples
    const studentSamplesQuery = query(collection(db,
collections.studentSamples), where("courseId", "==", course.id));
```

```
    onSnapshot(studentSamplesQuery, (snapshot) => {
      setStudentSamples(snapshot.docs.map(doc => ({ id: doc.id,
...doc.data() })));
    }, (error) => {
      console.error("Error fetching student samples:", error);
      setInfoModalMessage('Failed to load student samples.');
      setShowInfoModal(true);
    });

    // Fetch Feedback Snippets
    const feedbackSnippetsQuery = query(collection(db,
collections.feedbackSnippets));
    onSnapshot(feedbackSnippetsQuery, (snapshot) => {
      setFeedbackSnippets(snapshot.docs.map(doc => ({ id: doc.id,
...doc.data() })));
    }, (error) => {
      console.error("Error fetching feedback snippets:", error);
      setInfoModalMessage('Failed to load feedback snippets.');
      setShowInfoModal(true);
    });


  }, [db, userId, isAuthReady, course?.id, collections.assignments,
collections.syllabi, collections.studentSamples,
collections.feedbackSnippets]);

  useEffect(() => {
    fetchData();
  }, [fetchData]);

  // General handler for adding items
  const handleAddItem = async (collectionName, data, successMessage)
=> {
    if (!db || !userId) {
      setInfoModalMessage('Database not ready. Please wait.');
      setShowInfoModal(true);
      return;
    }
    try {
      await addDoc(collection(db, collections[collectionName]), data);
      setInfoModalMessage(successMessage);
      setShowInfoModal(true);
      // Close forms after successful add
      setShowAddAssignmentForm(false);
      setShowAddSyllabusForm(false);
      setShowAddSampleForm(false);
    } catch (e) {
      console.error(`Error adding ${collectionName}: `, e);
```

```
      setInfoModalMessage(`Error adding ${collectionName.slice(0,
-1)}. Please try again.`);
      setShowInfoModal(true);
    }
  };

  // General handler for updating items
  const handleUpdateItem = async (collectionName, id, data,
successMessage) => {
    if (!db || !userId) {
      setInfoModalMessage('Database not ready. Please wait.');
      setShowInfoModal(true);
      return;
    }
    try {
      const docRef = doc(db, collections[collectionName], id);
      await updateDoc(docRef, data);
      setInfoModalMessage(successMessage);
      setShowInfoModal(true);
      // Reset editing state after successful update
      setEditingAssignment(null);
      setEditingSyllabus(null);
      setEditingSample(null);
    } catch (e) {
      console.error(`Error updating ${collectionName}: `, e);
      setInfoModalMessage(`Error updating ${collectionName.slice(0,
-1)}. Please try again.`);
      setShowInfoModal(true);
    }
  };

  // General handler for deleting items
  const handleDeleteItem = async () => {
    if (!db || !userId || !itemToDelete || !deleteType || !storage) {
      setInfoModalMessage('Database, item data, or storage not ready.
Please wait.');
      setShowInfoModal(true);
      return;
    }
    try {
      // Check for and delete associated files before deleting
document
      if (itemToDelete.fileUrl &&
itemToDelete.fileUrl.startsWith('https://firebasestorage.googleapis.co
m/')) {
        await deleteFileFromFirebase(itemToDelete.fileUrl, storage,
setInfoModalMessage, setShowInfoModal);
      }
```

```
      if (itemToDelete.instructionsFileUrl &&
itemToDelete.instructionsFileUrl.startsWith('https://firebasestorage.g
oogleapis.com/')) {
        await deleteFileFromFirebase(itemToDelete.instructionsFileUrl,
storage, setInfoModalMessage, setShowInfoModal);
      }
      if (itemToDelete.rubricFileUrl &&
itemToDelete.rubricFileUrl.startsWith('https://firebasestorage.googlea
pis.com/')) {
        await deleteFileFromFirebase(itemToDelete.rubricFileUrl,
storage, setInfoModalMessage, setShowInfoModal);
      }
      if (itemToDelete.sampleFileUrl &&
itemToDelete.sampleFileUrl.startsWith('https://firebasestorage.googlea
pis.com/')) {
        await deleteFileFromFirebase(itemToDelete.sampleFileUrl,
storage, setInfoModalMessage, setShowInfoModal);
      }

      const docRef = doc(db, collections[deleteType],
itemToDelete.id);
      await deleteDoc(docRef);
      setInfoModalMessage(`${deleteType.slice(0, -1)} deleted
successfully!`);
      setShowInfoModal(true);
    } catch (e) {
      console.error(`Error deleting ${deleteType}: `, e);
      setInfoModalMessage(`Error deleting ${deleteType.slice(0, -1)}.
Please try again.`);
      setShowInfoModal(true);
    } finally {
      setShowConfirmModal(false);
      setItemToDelete(null);
      setDeleteType('');
    }
  };

  // Confirm deletion
  const confirmDeleteItem = (item, type) => {
    setItemToDelete(item);
    setDeleteType(type);
    setShowConfirmModal(true);
  };

  return (
    <div className="p-4 sm:p-6 lg:p-8 flex-grow">
      <button
        onClick={onBack}
```

```
        className="mb-6 inline-flex items-center bg-gray-200
hover:bg-gray-300 text-gray-800 font-bold py-2 px-4 rounded-lg
transition duration-200 ease-in-out focus:outline-none focus:ring-2
focus:ring-gray-400 focus:ring-opacity-75"
      >
        <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5
mr-2" viewBox="0 0 20 20" fill="currentColor">
          <path fillRule="evenodd" d="M12.707 5.293a1 1 0 010
1.414L9.414 10l3.293 3.293a1 1 0 01-1.414 1.414l-4-4a1 1 0
010-1.414l4-4a1 1 0 011.414 0z" clipRule="evenodd" />
        </svg>
        Back to Courses
      </button>

      <h1 className="text-3xl sm:text-4xl font-bold text-gray-800 mb-8
text-center">{course.name}</h1>
      <p className="text-gray-600 text-lg text-center
mb-10">{course.description}</p>

      {/* Quick Actions for Feedback */}
      <div className="flex justify-center mb-8 space-x-4">
        <button
          onClick={() => setShowFeedbackManager(true)}
          className="bg-purple-600 hover:bg-purple-700 text-white
font-bold py-2 px-5 rounded-lg shadow-md transition duration-200
ease-in-out transform hover:scale-105 focus:outline-none focus:ring-2
focus:ring-purple-500 focus:ring-opacity-75"
        >
          Manage Feedback Snippets
        </button>
        <button
          onClick={() => setShowFeedbackInsights(true)}
          className="bg-indigo-600 hover:bg-indigo-700 text-white
font-bold py-2 px-5 rounded-lg shadow-md transition duration-200
ease-in-out transform hover:scale-105 focus:outline-none focus:ring-2
focus:ring-indigo-500 focus:ring-opacity-75"
        >
          View Feedback Insights
        </button>
      </div>


      {/* Assignments Section */}
      <section className="mb-12 bg-white p-6 rounded-lg shadow-md">
        <div className="flex justify-between items-center mb-6">
          <h2 className="text-2xl font-semibold
text-gray-800">Assignments</h2>
          <button
```

```
            onClick={() => setShowAddAssignmentForm(true)}
            className="bg-indigo-600 hover:bg-indigo-700 text-white
font-bold py-2 px-4 rounded-lg transition duration-200 shadow"
          >
            Add Assignment
          </button>
        </div>
        {assignments.length === 0 ? (
          <p className="text-gray-600">No assignments added yet.</p>
        ) : (
          <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
            {assignments.map(assignment => (
              <div key={assignment.id} className="bg-gray-50 p-4
rounded-lg border border-gray-200 shadow-sm flex flex-col">
                <h3 className="text-lg font-semibold text-gray-800
mb-2">{assignment.name}</h3>
                <p className="text-gray-600 text-sm mb-2">Due:
{assignment.dueDate}</p>

                <div className="mb-2">
                  <p className="font-medium
text-gray-700">Instructions:</p>
                  {assignment.instructionsFileUrl ? (
                    <a href={assignment.instructionsFileUrl}
target="_blank" rel="noopener noreferrer" className="text-blue-500
hover:underline text-sm block break-all">
                      View Instructions PDF
                    </a>
                  ) : (
                    <p className="text-gray-700 text-sm
line-clamp-3">{assignment.instructions || 'No instructions
provided.'}</p>
                  )}
                </div>

                <div className="flex-grow">
                  <p className="font-medium text-gray-700">Rubric:</p>
                  {assignment.rubricFileUrl ? (
                    <a href={assignment.rubricFileUrl} target="_blank"
rel="noopener noreferrer" className="text-blue-500 hover:underline
text-sm block break-all">
                      View Rubric PDF
                    </a>
                  ) : assignment.rubricData ? (
                    <div className="text-sm border border-gray-200
rounded-md p-2 mt-1 bg-white max-h-40 overflow-y-auto">
                      <RubricDisplay
rubric={JSON.parse(assignment.rubricData)} />
```

```
                    </div>
                  ) : (
                    <p className="text-gray-700 text-sm">No rubric
provided.</p>
                  )}
                </div>

                {assignment.quizQuestions &&
JSON.parse(assignment.quizQuestions).length > 0 && (
                    <div className="flex-grow mt-2">
                        <p className="font-medium text-gray-700">Quiz
Questions:</p>
                        <div className="text-sm border border-gray-200
rounded-md p-2 mt-1 bg-white max-h-40 overflow-y-auto">

{JSON.parse(assignment.quizQuestions).map((q, qIndex) => (
                                <div key={qIndex} className="mb-2">
                                    <p className="font-medium
text-gray-900">{qIndex + 1}. {q.questionText}</p>
                                    <ul className="list-disc
list-inside ml-4">
                                        {q.options.map((option,
oIndex) => (
                                            <li key={oIndex}
className={option.isCorrect ? 'text-green-700 font-semibold' :
'text-gray-700'}>
                                                {option.text}
{option.isCorrect && '(Correct)'}
                                            </li>
                                        ))}
                                    </ul>
                                </div>
                            ))}
                        </div>
                    </div>
                )}

                <div className="flex justify-end space-x-2 mt-3">
                  <button
                    onClick={() => setEditingAssignment(assignment)}
                    className="bg-yellow-500 hover:bg-yellow-600
text-white text-xs font-medium py-1.5 px-3 rounded-lg"
                  >
                    Edit
                  </button>
                  <button
                    onClick={() => confirmDeleteItem(assignment,
'assignments')}
```

```
                    className="bg-red-500 hover:bg-red-600 text-white
text-xs font-medium py-1.5 px-3 rounded-lg"
                  >
                    Delete
                  </button>
                </div>
              </div>
            ))}
          </div>
        )}
      </section>

      {/* Syllabi Section */}
      <section className="mb-12 bg-white p-6 rounded-lg shadow-md">
        <div className="flex justify-between items-center mb-6">
          <h2 className="text-2xl font-semibold
text-gray-800">Syllabi</h2>
          <button
            onClick={() => setShowAddSyllabusForm(true)}
            className="bg-indigo-600 hover:bg-indigo-700 text-white
font-bold py-2 px-4 rounded-lg transition duration-200 shadow"
          >
            Add Syllabus
          </button>
        </div>
        {syllabi.length === 0 ? (
          <p className="text-gray-600">No syllabi added yet.</p>
        ) : (
          <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
            {syllabi.map(syllabus => (
              <div key={syllabus.id} className="bg-gray-50 p-4
rounded-lg border border-gray-200 shadow-sm">
                <h3 className="text-lg font-semibold
text-gray-800">{syllabus.title}</h3>
                {syllabus.fileUrl ? (
                  <p className="text-gray-700 text-sm mb-2">File: <a
href={syllabus.fileUrl} target="_blank" rel="noopener noreferrer"
className="text-blue-500 hover:underline break-all">View Syllabus
PDF</a></p>
                ) : (
                  <p className="text-gray-700 text-sm
line-clamp-3">{syllabus.content || 'No text content provided.'}</p>
                )}
                <div className="flex justify-end space-x-2 mt-3">
                  <button
                    onClick={() => setEditingSyllabus(syllabus)}
                    className="bg-yellow-500 hover:bg-yellow-600
text-white text-xs font-medium py-1.5 px-3 rounded-lg"
```

```
                  >
                    Edit
                  </button>
                  <button
                    onClick={() => confirmDeleteItem(syllabus,
'syllabi')}
                    className="bg-red-500 hover:bg-red-600 text-white
text-xs font-medium py-1.5 px-3 rounded-lg"
                  >
                    Delete
                  </button>
                </div>
              </div>
            ))}
          </div>
        )}
      </section>

      {/* Student Samples Section */}
      <section className="mb-12 bg-white p-6 rounded-lg shadow-md">
        <div className="flex justify-between items-center mb-6">
          <h2 className="text-2xl font-semibold text-gray-800">Student
Samples & Feedback</h2>
          <button
            onClick={() => setShowAddSampleForm(true)}
            className="bg-indigo-600 hover:bg-indigo-700 text-white
font-bold py-2 px-4 rounded-lg transition duration-200 shadow"
          >
            Add Student Sample
          </button>
        </div>
        {studentSamples.length === 0 ? (
          <p className="text-gray-600">No student samples added
yet.</p>
        ) : (
          <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
            {studentSamples.map(sample => (
              <div key={sample.id} className="bg-gray-50 p-4
rounded-lg border border-gray-200 shadow-sm flex flex-col">
                <h3 className="text-lg font-semibold
text-gray-800">{sample.studentName}</h3>
                <p className="text-gray-600 text-sm">Assignment:
{assignments.find(a => a.id === sample.assignmentId)?.name ||
'N/A'}</p>

                <div className="mb-2 flex-grow">
                  <p className="font-medium text-gray-700">Sample:</p>
                  {sample.sampleFileUrl ? (
```

```
                    <a href={sample.sampleFileUrl} target="_blank"
rel="noopener noreferrer" className="text-blue-500 hover:underline
text-sm block break-all">
                      View Student Sample
                    </a>
                  ) : (
                    <p className="text-gray-700 text-sm
line-clamp-3">{sample.sampleContent || 'No sample content
provided.'}</p>
                  )}
                </div>

                <p className="text-gray-700 text-sm
line-clamp-3">Feedback: {sample.feedback || 'No feedback yet.'}</p>
                <p className="text-gray-700 font-medium">Grade:
{sample.grade}</p>
                <div className="flex justify-end space-x-2 mt-3">
                  <button
                    onClick={() => setEditingSample(sample)}
                    className="bg-yellow-500 hover:bg-yellow-600
text-white text-xs font-medium py-1.5 px-3 rounded-lg"
                  >
                    Edit
                  </button>
                  <button
                    onClick={() => confirmDeleteItem(sample,
'studentSamples')}
                    className="bg-red-500 hover:bg-red-600 text-white
text-xs font-medium py-1.5 px-3 rounded-lg"
                  >
                    Delete
                  </button>
                </div>
              </div>
            ))}
          </div>
        )}
      </section>

      {/* Modals for forms */}
      {showAddAssignmentForm && (
        <AssignmentForm
          courseId={course.id}
          courseName={course.name} // Pass courseName for AI prompt
          storage={storage}
          onSave={(data) => handleAddItem('assignments', data,
'Assignment added successfully!')}
          onCancel={() => setShowAddAssignmentForm(false)}
```

```
        />
      )}
      {editingAssignment && (
        <AssignmentForm
          courseId={course.id}
          courseName={course.name} // Pass courseName for AI prompt
          storage={storage}
          initialAssignmentData={editingAssignment}
          onSave={(data) => handleUpdateItem('assignments',
editingAssignment.id, data, 'Assignment updated successfully!')}
          onCancel={() => setEditingAssignment(null)}
        />
      )}

      {showAddSyllabusForm && (
        <SyllabusForm
          courseId={course.id}
          storage={storage}
          onSave={(data) => handleAddItem('syllabi', data, 'Syllabus
added successfully!')}
          onCancel={() => setShowAddSyllabusForm(false)}
        />
      )}
      {editingSyllabus && (
        <SyllabusForm
          courseId={course.id}
          storage={storage}
          initialSyllabusData={editingSyllabus}
          onSave={(data) => handleUpdateItem('syllabi',
editingSyllabus.id, data, 'Syllabus updated successfully!')}
          onCancel={() => setEditingSyllabus(null)}
        />
      )}

      {showAddSampleForm && (
        <StudentSampleForm
          courseId={course.id}
          allAssignments={assignments} // Pass all assignments to
student sample form
          storage={storage}
          feedbackSnippets={feedbackSnippets} // Pass feedback
snippets to sample form
          onSave={(data) => handleAddItem('studentSamples', data,
'Student sample added successfully!')}
          onCancel={() => setShowAddSampleForm(false)}
        />
      )}
      {editingSample && (
```

```
        <StudentSampleForm
          courseId={course.id}
          allAssignments={assignments} // Pass all assignments to
student sample form
          storage={storage}
          feedbackSnippets={feedbackSnippets} // Pass feedback
snippets to sample form
          initialSampleData={editingSample}
          onSave={(data) => handleUpdateItem('studentSamples',
editingSample.id, data, 'Student sample updated successfully!')}
          onCancel={() => setEditingSample(null)}
        />
      )}

      {showConfirmModal && (
        <ConfirmModal
          message={`Are you sure you want to delete this
${deleteType.slice(0, -1)}?`}
          onConfirm={handleDeleteItem}
          onCancel={() => setShowConfirmModal(false)}
        />
      )}

      {showInfoModal && <InfoModal message={infoModalMessage}
onClose={() => setShowInfoModal(false)} />}
      {showFeedbackManager && <FeedbackSnippetsManager onClose={() =>
setShowFeedbackManager(false)} />}
      {showFeedbackInsights && <FeedbackInsights
allStudentSamples={studentSamples} allAssignments={assignments}
onClose={() => setShowFeedbackInsights(false)} />}
    </div>
  );
};

// --- Main App Component ---

function App() {
  const [db, setDb] = useState(null);
  const [auth, setAuth] = useState(null);
  const [storage, setStorage] = useState(null);
  const [userId, setUserId] = useState(null);
  const [isAuthReady, setIsAuthReady] = useState(false);
  const [selectedCourse, setSelectedCourse] = useState(null);
  const [showInfoModal, setShowInfoModal] = useState(false);
  const [infoModalMessage, setInfoModalMessage] = useState('');

  // Initialize Firebase and handle authentication
  useEffect(() => {
```

```
    const appId = typeof __app_id !== 'undefined' ? __app_id :
'default-app-id';
    const firebaseConfig = typeof __firebase_config !== 'undefined' ?
JSON.parse(__firebase_config) : {};

    if (Object.keys(firebaseConfig).length === 0) {
      console.error("Firebase config is missing or empty.");
      setInfoModalMessage('Firebase configuration is missing. Please
ensure it is correctly set up.');
      setShowInfoModal(true);
      // Mark auth as ready so the app doesn't stay in loading state
forever if config is missing
      setIsAuthReady(true);
      return;
    }

    let app;
    let firestore;
    let authInstance;
    let storageInstance;

    try {
      app = initializeApp(firebaseConfig);
      firestore = getFirestore(app);
      authInstance = getAuth(app);
      storageInstance = getStorage(app);

      setDb(firestore);
      setAuth(authInstance);
      setStorage(storageInstance);

      // Attempt initial sign-in.
      const initialSignIn = async () => {
        const initialAuthToken = typeof __initial_auth_token !==
'undefined' ? __initial_auth_token : null;
        if (initialAuthToken) {
          try {
            await signInWithCustomToken(authInstance,
initialAuthToken);
          } catch (error) {
            console.error("Error with custom token sign-in:", error);
            // If custom token fails, try anonymous sign-in as a
fallback
            try {
                await signInAnonymously(authInstance);
            } catch (anonError) {
                console.error("Error with anonymous sign-in
fallback:", anonError);
```

```
                setInfoModalMessage(`Authentication failed:
${error.message}. Please try refreshing.`);
                setShowInfoModal(true);
              }
            }
          } else {
            // If no custom token is provided, proceed directly with
anonymous sign-in
            try {
              await signInAnonymously(authInstance);
            } catch (error) {
              console.error("Error with anonymous sign-in:", error);
              setInfoModalMessage(`Authentication failed:
${error.message}. Please try refreshing.`);
              setShowInfoModal(true);
            }
          }
        };
        initialSignIn(); // Call it immediately

        // Listener for auth state changes - this will set userId and
isAuthReady
        const unsubscribe = onAuthStateChanged(authInstance, (user) => {
          if (user) {
            setUserId(user.uid);
          } else {
            setUserId(null); // Clear userId if logged out
          }
          setIsAuthReady(true); // Mark ready after initial auth check
is complete
        });

        return () => unsubscribe(); // Cleanup auth listener on
component unmount
      } catch (error) {
        console.error("Error initializing Firebase:", error);
        setInfoModalMessage(`Error initializing Firebase:
${error.message}.`);
        setShowInfoModal(true);
        setIsAuthReady(true); // Mark ready even on init error to avoid
infinite loading
      }
    }, []); // Empty dependency array means this runs once on mount

    if (!isAuthReady) {
      return (
        <div className="flex items-center justify-center min-h-screen
bg-gray-100 font-sans">
```

```
        <div className="text-center text-lg text-gray-700">Loading
application...</div>
      </div>
    );
  }

  return (
    // Pass storage instance through context
    <FirebaseContext.Provider value={{ db, auth, storage, userId,
isAuthReady }}>
      <div className="min-h-screen bg-gray-100 font-sans text-gray-800
flex flex-col">
        <header className="bg-white shadow-sm py-4 px-6 md:px-8
lg:px-10 flex justify-between items-center">
          <h1 className="text-xl md:text-2xl font-bold
text-blue-700">Teacher Dashboard</h1>
          {userId && (
            <div className="text-sm text-gray-600 hidden sm:block">
              User ID: <span className="font-mono text-xs
break-all">{userId}</span>
            </div>
          )}
        </header>

        <main className="flex-grow flex flex-col items-center
justify-center p-4">
          {selectedCourse ? (
            <CourseDetail course={selectedCourse} onBack={() =>
setSelectedCourse(null)} />
          ) : (
            <CourseList onSelectCourse={setSelectedCourse} />
          )}
        </main>

        {showInfoModal && <InfoModal message={infoModalMessage}
onClose={() => setShowInfoModal(false)} />}
      </div>
    </FirebaseContext.Provider>
  );
}

export default App;
```