TEST WORD FILE

Code dump

/\*

Okay so this is a remake of my fileUpload, the problem I had with the previous one was that I did not fully implement, the upload part into the component itself. I tried

to retrieve the files from the component calling fileuplaod, this created a couple of issue like this error, "cant updata a component state from another component",

another problem was I was having issues clearing files in the fileupload component, when I needed them cleared, this problem has lead me to recreate a different

file Upload component. I plan to use this uplaod with the file, moodboard, call sheet and budget&receipts components. I am sure having my new file upload

handle the processing and uploading of files to server on its own will resolve a lot of the headaches I have been experiencing.

P.S. I will still keep the previous file uplaod for the new projectform, the project form is what caused this while issue in the first place.

But after implementing this process I will try and see if I can change the new project

upload component to this one.

P.S. This fileuploadrebuild will also compress all images

TODO

\*/

import React, { useState, useRef, useEffect } from "react";

import { Upload } from "react-feather";

import imageCompression from "browser-image-compression";

import Button from "../../UIElements/Button/Button";

import dragDrop from "../../../Images/dragdrop.png";

import trashCan from "../../../Images/trashCan.png";

import "../FileUpload/FileUpload.css";

const FileUploadRebuild = (props) => {

    const [selectedFiles, setSelectedFiles] = useState([]); //will hold all upload files to display

    const [errorMessage, setErrorMessage] = useState(""); //string to display error message

    const [validFiles, setValidFiles] = useState([]); //this will hold validFiles, and will not allow duplicates using the useEffect below

    const [unsupportedFiles, setUnsupportedFiles] = useState([]);

    useEffect(() => {

        //using reduce, find, and concat methods to remove duplicates and add the individual values into the new array validFiles.

        let filteredArray = selectedFiles.reduce((file, current) => {

            const x = file.find((item) => item.name === current.name);

            if (!x) {

                return file.concat([current]);

            } else {

                return file;

            }

        }, []);

        setValidFiles([...filteredArray]);

    }, [selectedFiles]);

    // When a file is dragged into a browser from the OS, the browser will attempt to open and display it by default.

    //If you want to allow a drop, you must prevent the default handling of the event handlers.

    const dragOver = (e) => {

        e.preventDefault();

    };

    const dragEnter = (e) => {

        e.preventDefault();

    };

    const dragLeave = (e) => {

        e.preventDefault();

    };

    const fileDrop = (e) => {

        e.preventDefault();

        //e.dataTransfer is a Data Transfer object that holds the data that is being dragged during a drag-and-drop operation.

        const files = e.dataTransfer.files;

        if (files.length) {

            handleFiles(files);

        }

    };

    //used to check if file is an image type

    const validTypes = [

        "image/jpeg",

        "image/jpg",

        "image/png",

        "image/gif",

        "image/x-icon",

        "file/pdf",

        "file/txt",

    ];

    const otherValidTypes = [

        "application/pdf",

        "text/plain",

        "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet",

    ]

    //based on validate result handle file will either add file to selectedFile Array, or display and error

    const handleFiles = (files) => {

        for (let i = 0; i < files.length; i++) {

            if (validateFile(files[i])) {

                // console.log(files[i]);

                //!before I save the file to SelectFiles array state, I want to compress the image

                //first check if file is an image

                if (validTypes.indexOf(files[i].type) !== -1) {

                    //now compress file using browser-image-compression package

                    handleImageCompression(files[i]);

                }

                //add acceptAll prop, to validate true for files in files, callsheet, budget and receipts, and add to selected array

                //TODO might add an otherValidTypes check above to filter for pdf, word, excel and txt files types

            } else if (props.acceptAll) {

                console.log(files[i]);

                setSelectedFiles((prevArray) => [...prevArray, files[i]]);

            } else {

                // add a new property called invalid

                files[i]["invalid"] = true;

                // add to the same array so we can display the name of the file

                setSelectedFiles((prevArray) => [...prevArray, files[i]]);

                // set error message

                setErrorMessage("File type not permitted");

                //Each invalid file dropped by the user will be added to the array.

                setUnsupportedFiles((prevArray) => [...prevArray, files[i]]);

            }

        }

    };

    //function used to compress images

    function handleImageCompression(imageFile) {

        // var imageFile = event.target.files[0];

        // console.log("originalFile instanceof Blob", imageFile instanceof Blob); // true

        // console.log(`originalFile size ${imageFile.size / 1024 / 1024} MB`);

        var options = {

            maxSizeMB: 1,

            maxWidthOrHeight: 1920,

            useWebWorker: true,

        };

        imageCompression(imageFile, options)

            .then(function (compressedFile) {

                // console.log(

                //  "compressedFile instanceof Blob",

                //  compressedFile instanceof Blob

                // ); // true

                // console.log(

                //  `compressedFile size ${compressedFile.size / 1024 / 1024} MB`

                // ); // smaller than maxSizeMB

                // return compressedFile; // write your own logic

                console.log(compressedFile);

                setSelectedFiles((prevArray) => [...prevArray, compressedFile]);

            })

            .catch(function (error) {

                console.log(error);

                // setSelectedFiles((prevArray) => [...prevArray, imageFile]);

                // set error message

                setErrorMessage("An error occured while compressing");

                //Each invalid file dropped by the user will be added to the array.

                setUnsupportedFiles((prevArray) => [...prevArray, imageFile]);

            });

    }

    //this validates files, I really dont need this for files, callsheet, budget and receipts but I need for moodboard to allow only pictures

    //\*and thats unfortunate because I cant compress pdf, word or ecxel files that might be too big, we will see how this affects the program

    //\*in the future, and look at alternative then

    const validateFile = (file) => {

        //if not an image return false

        if (validTypes.indexOf(file.type) === -1) {

            return false;

        }

        // else if (props.acceptAll) {

        //  // setSelectedFiles((prevArray) => [...prevArray, file]);

        //  return true;

        // }

        //return true if file is an image

        return true;

    };

    //this method calculates the filesize and is used for display purposes

    //We return the string 0 Bytes if the size is zero. 1 KB is equivalent to 1024 bytes.

    const fileSize = (size) => {

        if (size === 0) return "0 Bytes";

        const k = 1024;

        const sizes = ["Bytes", "KB", "MB", "GB", "TB"];

        const i = Math.floor(Math.log(size) / Math.log(k));

        return parseFloat((size / Math.pow(k, i)).toFixed(2)) + " " + sizes[i];

    };

    //method for getting the file type from the file name.

    //The substring() method returns the characters after the . in the file name.

    const fileType = (fileName) => {

        return (

            fileName.substring(fileName.lastIndexOf(".") + 1, fileName.length) ||

            fileName

        );

    };

    //will remove file (delete button)

    const removeFile = (name) => {

        // find the index of the item

        // remove the item from array

        const validFileIndex = validFiles.findIndex((e) => e.name === name);

        validFiles.splice(validFileIndex, 1);

        // update validFiles array

        setValidFiles([...validFiles]);

        const selectedFileIndex = selectedFiles.findIndex((e) => e.name === name);

        selectedFiles.splice(selectedFileIndex, 1);

        // update selectedFiles array

        setSelectedFiles([...selectedFiles]);

        //Each invalid file dropped by the user will be removed from unsupportedFiles array.

        const unsupportedFileIndex = unsupportedFiles.findIndex(

            (e) => e.name === name

        );

        if (unsupportedFileIndex !== -1) {

            unsupportedFiles.splice(unsupportedFileIndex, 1);

            // update unsupportedFiles array

            setUnsupportedFiles([...unsupportedFiles]);

        }

    };

    //HANDLING CLICK EVENT FOR FILE UPLOAD

    //send validFiles and unsupportedFiles array to NewProjectForm

    // const { allFiles } = props;

    //set a useref hook

    const filePickerRef = useRef();

    // create a method pickImageHandler to register click event

    const pickImageHandler = () => {

        filePickerRef.current.click();

    };

    //add a method filesSelected to obtain selected files and pass it to the handleFile method

    const filesSelected = () => {

        if (filePickerRef.current.validFiles.length) {

            handleFiles(filePickerRef.current.validFiles);

        }

    };

    //method to handleUpload based on criteria(uploadFiles, uploadBudgetReceipt, uploadCallSheet, uploadMoodBoard)

    const handleUpload = () => {

        //will process and upload based on criteria

        //will send to project update action

        //display errors, if any make sure i dont lose file upload progress

        console.log(selectedFiles);

        //will have a switch that will upload to files, budget, callsheet, moodboard

        props.uploadFiles(selectedFiles);

        setTimeout(() => {

            validFiles.length = 0;

            setValidFiles([...validFiles]);

            setSelectedFiles([...validFiles]);

            setUnsupportedFiles([...validFiles]);

        }, [1000]);

        // //or i can use a settimeout to clear files

        // if (props.uploadSucessful) {

        //  console.log("Deleting...");

        //  validFiles.length = 0;

        //  setValidFiles([...validFiles]);

        //  setSelectedFiles([...validFiles]);

        //  setUnsupportedFiles([...validFiles]);

        // }

    };

    // const triggerFunc = () => {

    //  allFiles(validFiles);

    // };

    return (

        <React.Fragment>

            {/\* {unsupportedFiles.length === 0 && validFiles.length

                ? triggerFunc()

                : null} \*/}

            <input

                type="file"

                // id={props.id}

                ref={filePickerRef}

                style={{ display: "none" }}

                multiple

                onChange={filesSelected}

            />

            <label

                htmlFor="projectFiles"

                className="dragdrop blue-button center"

                onClick={pickImageHandler}

                // When a file is dragged into a browser from the OS, the browser will attempt to open and display it by default.

                //If you want to allow a drop, you must prevent the default handling of the event handlers.

                onDragOver={dragOver}

                onDragEnter={dragEnter}

                onDragLeave={dragLeave}

                onDrop={fileDrop}>

                <React.Fragment>

                    <img className="dragDropImg" src={dragDrop} alt="dragDropImg"></img>

                    <p>DRAG AND DROP OR CLICK TO SELECT FILE(S)</p>

                </React.Fragment>

            </label>

            <div className="file-display-container center">

                {/\* display all validFiles by looping thru selectedFIles array \*/}

                {validFiles.map((data, i) => (

                    <div className="file-status-bar" key={i}>

                        <div>

                            {/\* show file type \*/}

                            <div className="file-type">{fileType(data.name)}</div>

                            {/\* show file name and if obecjt has the invalid property style file-error\*/}

                            <span className={`file-name ${data.invalid ? "file-error" : ""}`}>

                                {data.name.length > 15

                                    ? data.name.slice(0, 15) + "..."

                                    : data.name}

                            </span>

                            {/\* Check whether the object contains the invalid property added, which would indicate an invalid file. Add the class name file-error. \*/}

                            <span className="file-size">({fileSize(data.size)})</span>{" "}

                            {data.invalid && (

                                <span className="file-error-message">({errorMessage})</span>

                            )}

                        </div>

                        {/\* used to trigger remove file function \*/}

                        <div className="file-remove" onClick={() => removeFile(data.name)}>

                            <img src={trashCan} alt="trashCan" />

                        </div>

                    </div>

                ))}

            </div>

            {/\* Display file counts \*/}

            <div style={{ paddingTop: "10px", paddingBottom: "10px" }}>

                {validFiles.length > 0 &&

                    (validFiles.length === 1

                        ? "1 File Selected"

                        : validFiles.length + " Files Selected")}

            </div>

            {/\* show submit button or error \*/}

            <div className="unsupported">

                {unsupportedFiles.length > 0 ? (

                    <div>

                        <h5>Please remove unsupported file(s)</h5>

                    </div>

                ) : validFiles.length > 0 ? (

                    <div className="center">

                        <Button type="submit" onClick={handleUpload}>

                            <Upload color="green" size={36} /> Upload

                        </Button>

                    </div>

                ) : null}

            </div>

        </React.Fragment>

    );

};

export default FileUploadRebuild;