import React, { useEffect, useState } from 'react';

import axios from 'axios';

import { getUser } from '../service/AuthService';

import '../index.css';

import './locationStyle.css';

import CircularProgress from '@mui/material/CircularProgress';

import Box from '@mui/material/Box';

import { LuAlarmClock } from "react-icons/lu";

import { useNavigate } from 'react-router-dom';

const addressUrl = 'https://lyg1apc3wl.execute-api.ap-southeast-2.amazonaws.com/prod/csvAddress';

const timeInUrl = 'https://lyg1apc3wl.execute-api.ap-southeast-2.amazonaws.com/prod/timein';

const timeOutUrl = 'https://lyg1apc3wl.execute-api.ap-southeast-2.amazonaws.com/prod/timeout';

localStorage.clear();

let jobControlTableName;

export default function LocationFinder() {

  // localStorage.setItem('userStreet', "not achieved yet");

  // localStorage.setItem('userSuburb', 'not achieved yet');

  const navigate = useNavigate();

  const user = getUser();

  const loginEmail = user !== 'undefined' && user ? user.email : '';

  const [status, setStatus] = useState('');

  const [message, setMessage] = useState('');

  const [isWorking, setIsWorking] = useState(false);

  const [csvData, setCsvData] = useState([]); //to store csv address

  const [callusername, setCallUsername] = useState(null);

  const [showClockButtons, setShowClockButtons] = useState(false);

  const [calendar, setCalendar] = useState(false);

  const [loading, setLoading] = useState(false); // New state for loading indicator

  const [isLocation, setIsLocation] = useState(false);

  const requestConfig = {

    headers: {

        'x-api-key': 'EmYB7EcYzn2NK1dUkD2kK8MA18r5dp6tQ7wB7U1d'

    }

  }

  useEffect(() => {

    // Make a GET request to your Lambda function's endpoint

    axios.get(addressUrl, requestConfig)

    .then((response) => {

      setCsvData(response.data);

      console.log('CSV Data:', response.data);

    })

    .catch((error) => {

      console.error(error);

    });

  }, [isWorking]); // This will run once when the component mounts

  let watchId;

  const findMyLocation = (isWorking) => {

    const success = (position) => {

      console.log(position);

      const latitude = position.coords.latitude;

      const longitude = position.coords.longitude;

      console.log(latitude + " " + longitude);

      const geoApiUrl = `https://nominatim.openstreetmap.org/reverse?format=json&lat=${latitude}&lon=${longitude}`

      fetch(geoApiUrl)

      .then((res) => res.json())

      .then((data) => {

        console.log(data);

        let matchFound = false;

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

          if(

            isWorking &&

            csvData[i]?.street  === data.address.road &&

            csvData[i]?.suburb === data.address.suburb

            ) {

            setStatus('')

            const timeIn = new Date().toLocaleString("en-AU", {

              timeZone: "Australia/Sydney",

            });

            const requestBodyTimeIn = {

              email: loginEmail,

              timeIn: timeIn

            }

            axios.post(timeInUrl, requestBodyTimeIn, requestConfig)

            .then(response => {

              const street = csvData[i]?.street

              const suburb = csvData[i]?.suburb

              //jobControlTableName = "Gladstone\_Parade\_Lindfield"

              jobControlTableName = street?.split(' ').join('\_') + '\_' + suburb?.split(' ').join('\_');

              console.log(jobControlTableName)

              window.localStorage.setItem('jobControlTableName', jobControlTableName);

              console.log("local storage " + window.localStorage.getItem('jobControlTableName', jobControlTableName));

              setStatus('You are working at:\n' + csvData[i].street + ' ' + csvData[i].suburb + '\n at ' + timeIn);

              setIsWorking(true);

              setIsLocation(true);

              setLoading(false);

              setCalendar(true);

              matchFound = true;

            }).catch(error => {

              if (error.response && (error.response.status === 401 || error.response.status === 403)) {

                  setMessage(error.response.data.message);

              }

              else {

                  setMessage('Sorry, backend server is down, please try again later')

              }

          })

            break;

          }

          else if (

            !isWorking &&

            csvData[i]?.street  === data.address.road &&

            csvData[i]?.suburb === data.address.suburb

          ) {

            setStatus('');

            const timeOut = new Date().toLocaleString("en-AU", {

              timeZone: "Australia/Sydney",

            });

            const requestBodyTimeOut = {

              email: loginEmail,

              timeOut: timeOut

            }

            axios.post(timeOutUrl, requestBodyTimeOut, requestConfig).then(response => {

              setStatus('You are finished at:' + timeOut);

              setIsWorking(false);

              setIsLocation(true);

              setLoading(false);

              matchFound = true;

            }).catch(error => {

              if (error.response && (error.response.status === 401 || error.response.status === 403)) {

                  setMessage(error.response.data.message);

              }

              else {

                  setMessage('Sorry, backend server is down, please try again later')

              }

          })

            break;

          }

          // else {

          //   setStatus('')

          //   setLoading(false);

          //   setStatus('Your GPS may be incorrectly tracked, please try again');

          //   break;

          // }

        }

        if (!isLocation) {

          setLoading(false);

          setStatus('Your GPS may be incorrectly tracked, please try again. Your current location: ' +data.address.road + ' ' + data.address.suburb);

        }

      })

      .catch((error) => {

        console.error('Error fetching data:', error);

      })

      //navigator.geolocation.clearWatch(watchId);

    }

    const errors = () => {

        //status.textContent = 'Unble to retrieve your location, please unblock the location permission in your browser'

        setStatus('Unble to retrieve your location, please unblock the location permission in your browser');

        //navigator.geolocation.clearWatch(watchId);

      }

      navigator.geolocation.getCurrentPosition(success, errors);

    };

    const handleClockin = () => {

        findMyLocation(true);

        setLoading(true);

        //setIsWorking(true);

      };

    const handleClockout = () => {

        findMyLocation(false);

        setLoading(true)

        //setIsWorking(false);

    }

    const punchClock = () => {

      setIsWorking(false);

      setCallUsername(loginEmail);

      console.log("Punch Clock clicked " + loginEmail);

      setShowClockButtons((prevShowClockButtons) => !prevShowClockButtons);

    }

    const handleCalendar = () => {

      navigate('/calendar')

    }

  return (

    <div>

      <button className="punch-button" onClick={punchClock}><LuAlarmClock /> Punch Clock</button>

        <div className={`clockin-clockout ${showClockButtons ? 'active' : 'inactive'}`}>

          <button className='clockin' onClick={handleClockin}>Clock-in</button>

          <button className='clockout' onClick={handleClockout}>Clock-out</button>

          {loading && (

                <Box sx={{ position: "fixed", top: "40%", left: "45%", zIndex: "1000" }}>

                  <CircularProgress />

                </Box>

            )}

          <p className="status">{status}</p>

          <div className={`calendar-button ${calendar ? 'active' : 'inactive'}`}>

              <button className='go-to-calendar-button' onClick={handleCalendar}>Go to Calendar</button>

          </div>

        </div>

    </div>

  );

};

const jobTableNameFunction = () => {

  if(jobControlTableName){

    console.log('this is the export function')

    return jobControlTableName;

  }

  return "some error!!!!"

}

export { jobTableNameFunction }

        const params = {

            TableName: tableName

        }

        dynamodb.scan(params, (err, data) => {

            if (err) {

                console.log("error: " + err);

                return util.buildResponse(401, {

                    message: "Error: " + err

                })

            } else {

                console.log("OK" + data.Items);

                return util.buildResponse(200, {

                    message: "OK " + data.Items

                })

            }

        })