I’m building a React App and here's the story so far

Current Setup Recap:

Smart meter > CAD monitor > Tablet running mosquitto > mqtt handler > supbase “Readings” table and Daily Energy Average table > front end BuildingDashboard.js file which will show the daily averages over time until we have enough data for kWh/m2/per annum

The tablet also auto-pulls updated code from GitHub and restarts the handler via pm2.

🔄 What We’re Doing Now

starting to bring in data for the other data fields. We now have a fixed external temperature reading and we’re now trying to pull in the other sensors from thingitude.

I’ve attached the relevant files… and getting ‘Cannot GET /api/getExternalTemperature’ when I test it on ‘http://localhost:5000/api/getExternalTemperature?lat=51.5074&lon=0.1278’… where am I going wrong?

BuildingDashboard.js:

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

import AnalogGauge from "../../components/AnalogGauge";

import supabase from "../../supabaseClient";

const BuildingDashboard = () => {

// 🔒 Helper: get boolean from localStorage safely

const getPersistentBoolean = (key, fallback = false) => {

const val = localStorage.getItem(key);

return val === null ? fallback : val === 'true';

};

const [buildingArea, setBuildingArea] = useState(() => {

const savedArea = localStorage.getItem('buildingArea');

return savedArea ? Number(savedArea) : 50;

});

const [location, setLocation] = useState(() => {

const savedLocation = localStorage.getItem('location');

return savedLocation ? JSON.parse(savedLocation) : null;

});

const [isAreaLocked, setIsAreaLocked] = useState(() => getPersistentBoolean('isAreaLocked'));

const [isLocationLocked, setIsLocationLocked] = useState(() => getPersistentBoolean('isLocationLocked'));

const [sensorData, setSensorData] = useState({

energyUse: 0,

temperature: 0,

externalTemp: 0,

humidity: 0,

co2: 0,

vocs: 0,

pm25: 0,

});

const [performanceValue, setPerformanceValue] = useState(0);

const [historicalPerformance, setHistoricalPerformance] = useState(0);

const [carbonCredits, setCarbonCredits] = useState(0);

// 🔁 Sync lock states to localStorage

useEffect(() => {

localStorage.setItem('isAreaLocked', isAreaLocked);

}, [isAreaLocked]);

useEffect(() => {

localStorage.setItem('isLocationLocked', isLocationLocked);

}, [isLocationLocked]);

// 🌡️ Fetch external temp based on saved location

const fetchExternalTemperature = async () => {

if (!location) {

console.log("No location saved yet, skipping external temp fetch.");

return;

}

try {

const response = await fetch(`http://localhost:5000/api/getExternalTemperature?lat=${location.latitude}&lon=${location.longitude}`);

const data = await response.json();

if (data && data.externalTemperature !== undefined) {

setSensorData(prev => ({ ...prev, externalTemp: data.externalTemperature }));

console.log("Fetched external temperature:", data.externalTemperature);

} else {

console.error("No externalTemperature in response:", data);

}

} catch (error) {

console.error("Error fetching external temperature:", error);

}

};

const fetchLongTermAverage = async () => {

try {

const { data, error } = await supabase

.from('DailyEnergyTotals')

.select('total\_energy\_kwh, day')

.order('day', { ascending: false })

.limit(1)

.single();

if (error) throw error;

if (data) {

const dailyTotalEnergy = data.total\_energy\_kwh || 0;

setHistoricalPerformance(dailyTotalEnergy);

console.log("Fetched historical performance (daily total energy):", dailyTotalEnergy);

} else {

console.log("No historical data found.");

}

} catch (err) {

console.error("Error fetching historical performance data:", err.message);

}

};

useEffect(() => {

fetchLongTermAverage();

}, []);

useEffect(() => {

if (historicalPerformance && buildingArea > 0) {

const energyPerSqM = historicalPerformance / buildingArea;

const invertedPerformance = energyPerSqM > 0 ? (1 / energyPerSqM) : 0;

const scaledPerformanceValue = Math.min(invertedPerformance \* 10, 100);

setPerformanceValue(scaledPerformanceValue);

console.log("Performance Value updated:", scaledPerformanceValue);

}

}, [historicalPerformance, buildingArea]);

useEffect(() => {

fetchExternalTemperature();

const interval = setInterval(fetchExternalTemperature, 60 \* 60 \* 1000); // refresh hourly

return () => clearInterval(interval);

}, [location]);

const handleAreaChange = (e) => {

if (isAreaLocked) return;

const newArea = Number(e.target.value);

setBuildingArea(newArea);

localStorage.setItem('buildingArea', newArea);

};

const handleAreaLockToggle = () => {

setIsAreaLocked(prev => !prev);

console.log("Toggled area lock");

};

const handleGeolocate = () => {

if (isLocationLocked) return;

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition(

({ coords }) => {

const newLocation = { latitude: coords.latitude, longitude: coords.longitude };

setLocation(newLocation);

localStorage.setItem('location', JSON.stringify(newLocation));

console.log(`📍 Latitude: ${coords.latitude}, Longitude: ${coords.longitude}`);

},

(err) => console.error("Geolocation error:", err)

);

}

};

const handleLocationLockToggle = () => {

setIsLocationLocked(prev => !prev);

console.log("Toggled location lock");

};

console.log("Sensor Data:", sensorData);

console.log("Performance Value:", performanceValue);

console.log("Historical Performance:", historicalPerformance);

return (

<div className="min-h-screen bg-white p-4 flex flex-col space-y-6">

{/\* Input Section \*/}

<div className="bg-gray-100 p-4 rounded shadow">

<h2 className="text-lg font-bold mb-2">Data Input</h2>

<div className="flex flex-wrap items-center gap-4 mb-4">

<button className="bg-green-500 text-white px-3 py-2 rounded">

Scan for Smart Meter

</button>

<button className="bg-green-500 text-white px-3 py-2 rounded">

Scan for Sensors

</button>

</div>

{/\* Building Area \*/}

<div className="flex items-center gap-2 mb-4">

<label className="font-semibold">Internal Area:</label>

<input

type="number"

className="border p-2 w-24"

value={buildingArea}

onChange={handleAreaChange}

disabled={isAreaLocked}

/>

<span>m²</span>

<button

onClick={handleAreaLockToggle}

className={`ml-2 px-2 py-1 rounded ${isAreaLocked ? 'bg-red-500' : 'bg-green-500'} text-white`}

>

{isAreaLocked ? "🔓 Unlock" : "🔒 Lock"}

</button>

</div>

{/\* Geolocation \*/}

<div className="flex items-center gap-2 mb-4">

<button

className={`px-3 py-2 rounded ${isLocationLocked ? 'bg-gray-400' : 'bg-green-500'} text-white`}

onClick={handleGeolocate}

disabled={isLocationLocked}

>

Geolocate

</button>

<button

onClick={handleLocationLockToggle}

className={`ml-2 px-2 py-1 rounded ${isLocationLocked ? 'bg-red-500' : 'bg-green-500'} text-white`}

>

{isLocationLocked ? "🔓 Unlock" : "🔒 Lock"}

</button>

</div>

{location && (

<div className="text-sm text-gray-600">

📍 Lat: {location.latitude.toFixed(5)}, Lng: {location.longitude.toFixed(5)}

</div>

)}

</div>

{/\* Performance Section \*/}

<div className="bg-gray-100 p-4 rounded shadow">

<h2 className="text-lg font-bold">Performance</h2>

<div className="flex items-center">

<AnalogGauge

value={performanceValue}

historicalValue={historicalPerformance}

/>

<div className="ml-4 text-sm">

<p>

<strong>Daily Average Energy Use:</strong> {historicalPerformance ? historicalPerformance.toFixed(4) : 'No Data'} kWh

</p>

<p>

<strong>Temperature:</strong> {sensorData.temperature !== null ? sensorData.temperature.toFixed(1) : 'No Data'} °C

</p>

<p>

<strong>External Temp:</strong> {sensorData.externalTemp !== null ? sensorData.externalTemp.toFixed(1) : 'No Data'} °C

</p>

<hr className="my-2" />

<p><strong>Humidity:</strong> {sensorData.humidity !== null ? sensorData.humidity.toFixed(1) : 'No Data'}%</p>

<p><strong>CO2:</strong> {sensorData.co2 !== null ? sensorData.co2.toFixed(1) : 'No Data'} ppm</p>

<p><strong>VOCs:</strong> {sensorData.vocs !== null ? sensorData.vocs.toFixed(2) : 'No Data'} ppm</p>

<p><strong>PM2.5:</strong> {sensorData.pm25 !== null ? sensorData.pm25.toFixed(1) : 'No Data'} µg/m³</p>

</div>

</div>

</div>

{/\* Carbon Credit Section \*/}

<div className="bg-gray-100 p-4 rounded shadow">

<h2 className="text-lg font-bold">Digital Carbon Credits</h2>

<p><strong>{carbonCredits}</strong> DCC</p>

<button className="bg-red-500 text-white px-4 py-2 w-32 rounded">

SELL CREDITS

</button>

</div>

</div>

);

};

export default BuildingDashboard;

Server.js:

const express = require('express');

const cors = require('cors');

const app = express();

const port = 5000;

require("dotenv").config();

const latestRoute = require('./routes/latest');

const historicalRoute = require('./routes/historical'); // ✅ ADD THIS LINE

const externalTemperatureRoute = require('./routes/externalTemperature');

app.use('/api/getExternalTemperature', externalTemperatureRoute);

app.use(cors());

app.use(express.json());

app.use('/latest', latestRoute);

app.use('/historical', historicalRoute); // ✅ Now this will work

app.listen(port, () => {

console.log(`Server running on port ${port}`);

});

externalTemperature.js

const express = require('express');

const router = express.Router();

const axios = require('axios');

require('dotenv').config();

router.get('/', async (req, res) => {

const { lat, lon } = req.query;

if (!lat || !lon) {

return res.status(400).json({ error: "Missing lat or lon parameters" });

}

try {

const apiKey = process.env.WBP; // Assuming your .env uses WBP

const response = await axios.get(

`https://api.openweathermap.org/data/2.5/weather`,

{

params: {

lat,

lon,

units: 'metric',

appid: apiKey,

}

}

);

const externalTemp = response.data.main.temp;

res.json({ externalTemperature: externalTemp });

} catch (error) {

console.error("Failed to fetch external temp:", error.message);

res.status(500).json({ error: "Failed to fetch external temperature" });

}

});

module.exports = router;