const handleCalculateDeforestationRate = async () => {

  try {

    const response = await fetch('http://localhost:8000/api/calculate\_four\_months\_before/calculate/', {

      method: 'POST',

      headers: {

        'Content-Type': 'application/json'

      },

      body: JSON.stringify({ date: startDate })

    });

    if (!response.ok) {

      throw new Error('Network response was not ok');

    }

    const data = await response.json();

    setCalculation(data.calculation);

    setResponseTableData(data); // Set the response data to be displayed in the table

    console.log(data); // Log the response data to the console

  } catch (error) {

    console.error('Error:', error);

  }

}

Old project identifier

                <Input

                type='text' // Use lowercase 'text' for text input

                size='md'

                placeholder='Project Identifier'

                value={project} // Assuming 'project' is the variable to save the input text

                onChange={(event) => setProject(event.target.value)} // Update the 'project' variable as the user types

                />

            {responseTableData && <ResponseTable responseData={responseTableData} />}

/ const handleDownloadShapefile = () => {

//     if (polygonCoordinates.length === 0) return;

//     const shapefileBuffer = generateShapefile(polygonCoordinates);

//     const blob = new Blob([shapefileBuffer], { type: 'application/zip' });

//     const url = URL.createObjectURL(blob);

//     const a = document.createElement('a');

//     a.href = url;

//     a.download = project + '.zip';

//     document.body.appendChild(a);

//     a.click();

//     document.body.removeChild(a);

//     URL.revokeObjectURL(url);

// };

// const generateShapefile = (coordinates) => {

//     // Ensure coordinates array is not empty

//     if (coordinates.length === 0) {

//         throw new Error("Coordinates array is empty");

//     }

//     // Generate features for the shapefile

//     const features = [{

//         type: 'Feature',

//         properties: {},

//         geometry: {

//             type: 'Polygon',

//             coordinates: [coordinates.map(coord => [coord.lng, coord.lat])]

//         }

//     }];

//     // Generate Shapefile buffer

//     const buffer = shpwrite.zip(features);

//     // Return the Shapefile buffer

//     return buffer;

// };

// const handleCalculateDeforestationRate = () => {

//     // Perform any necessary data validation...

//     // Assuming you have the KML content stored in a variable named kmlContent

//     //const kmlContent = generateKML(polygonCoordinates);

//     const startdate = startDate

//     axios.post('/utils\_sat.py', { startdate })

//       .then(response => {

//         // Handle the response from the server...

//         console.log(response.data);

//       })

//       .catch(error => {

//         // Handle any errors...

//         console.error('Error:', error);

//       });

//   };

//   async function handleCalculateDeforestationRate() {

//     const response = await fetch('http://localhost:8000/api/calculate\_four\_months\_before/', {

//         method: 'POST',

//         headers: {

//             'Content-Type': 'application/json'

//         },

//         body: JSON.stringify({ date: startDate })

//     });

//     const data = await response.json();

//     setCalculation(data.calculation);

// }

  const isPointWithinKmlLayer = async (point) => {

    // Check if the given point (latLng) is within the boundaries of the KML layer for the selected project

    if (project === 'Kayapo') {

      const kayapoKmlBounds = await getKmlLayerBounds('https://drive.google.com/uc?id=16c\_-xJIaRtq31VMGgckB3rlbkxuV7S8t');

      return kayapoKmlBounds && kayapoKmlBounds.contains(point);

    } else if (project === 'Yanomami') {

      const yanomamiKmlBounds = await getKmlLayerBounds('https://gateway.pinata.cloud/ipfs/QmXQDaAk6RkmDWuefL6H4EGJp9F9hpoRq6BkfqWaR75XPm');

      return yanomamiKmlBounds && yanomamiKmlBounds.contains(point);

    }

    // Default to true if no project is selected or if bounds checking logic is not implemented

    return true;

  };

  const getKmlLayerBounds = async (kmlUrl) => {

    return new Promise((resolve, reject) => {

      window.google.maps.event.addListenerOnce(kmlLayer, 'status\_changed', () => {

        if (kmlLayer.getStatus() === window.google.maps.KmlLayerStatus.OK) {

          const bounds = new window.google.maps.LatLngBounds();

          kmlLayer.getDefaultViewport().forEachLatLng((latLng) => {

            bounds.extend(latLng);

          });

          resolve(bounds);

        } else {

          reject('Failed to load KML layer.');

        }

      });

    });

  };

  const handleMapClick = (event) => {

    const point = new window.google.maps.LatLng(event.latLng.lat(), event.latLng.lng());

    // Check if the point is within the KML layer for the selected project

    isPointWithinKmlLayer(point)

      .then((isWithin) => {

        if (isWithin) {

          // Calculate the next zIndex for the polygon based on the current number of coordinates

          const nextZIndex = polygonCoordinates.length + 1;

          setPolygonCoordinates((prevCoordinates) => [

            ...prevCoordinates,

            {

              lat: event.latLng.lat(),

              lng: event.latLng.lng(),

              zIndex: nextZIndex, // Set zIndex for the polygon

            },

          ]);

        } else {

          console.log('Point is outside the KML layer boundaries.');

          // Optionally provide user feedback or handle the out-of-bounds scenario

        }

      })

      .catch((error) => {

        console.error('Error checking point within KML layer:', error);

        // Handle error scenario if needed

      });

  };

/\*   const loadKmlLayerKayapo = (mapInstance) => {

    const kmlUrl = "https://drive.google.com/uc?export=download&id=1N-ixnCzi4r0RWSro3OXz20ZUzGLWL\_DS";

    const kmlLayer = new window.google.maps.KmlLayer({

      url: kmlUrl,

      map:mapInstance,

      suppressInfoWindows: false,

      preserveViewport: true,

      zIndex: 0, // Lower zIndex to be below polygons

    });

    setkmlLayer(kmlLayer);

    kmlLayer.set('options', {

        preserveViewport: true,

        suppressInfoWindows: true,

        clickable: true,

        zIndex: 0,

        strokeColor: '#00FF00', // Green color

        strokeOpacity: 0.8,

        strokeWeight: 2,

        fillColor: '#00FF00', // Green color

        fillOpacity: 0.35,

      });

  }; \*/

Package.json

{ "private": true, "scripts": { "build": "next build", "dev": "next dev", "prettier": "prettier --write --ignore-unknown .", "prettier

": "prettier --check --ignore-unknown .", "start": "next start", "seed": "node -r dotenv/config ./scripts/seed.js" "custom-script": "./start\_dev.sh", "dev": "concurrently "npm run next-dev" "npm run custom-script"" },"

#!/bin/bash

# Function to close ports in the range 3000 to 3010

close\_ports\_range() {

for ((port=3000; port<=3010; port++)); do

echo "Closing port $port..."

# Find the process ID (PID) using the specified port and then terminate the process

pid=$(netstat -tuln | awk '{print $4}' | grep ":$port" | awk -F'/' '{print $1}' | awk -F'.' '{print $NF}')

if [ -n "$pid" ]; then

kill -9 $pid

fi

done

}

# Call the function to close the ports

close\_ports\_range

# Rest of your script...