import { Component, OnInit } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Division } from '../mapview.service';

import { Circle } from '../mapview.service';

import { DistributionCenter, MapviewService } from '../mapview.service';

import { Feeder11Service } from '../feeder11.service';

import \* as L from 'leaflet';

import { forkJoin } from 'rxjs';

@Component({

  selector: 'app-test',

  templateUrl: './test.component.html',

  styleUrls: ['./test.component.css']

})

export class TestComponent implements OnInit {

  regionDisplayMap: { [key: string]: string } = {

    '1': 'Indore',

    '2': 'Ujjain'

  };

  regions = ['1', '2'];

  circles: Circle[] = [];

  divisions: Division[] = [];

  dcs: { name: string, code: string }[] = [];

  selectedRegion = '';

  selectedCircle = '';

  selectedDivision = '';

  selectedDC = '';

  summary: any[] = [];

  feederCounts: any;

  feederDashboardData: any[];

  private map: L.Map | undefined;

  private markers: L.Marker[] = [];

  constructor(private mapview: MapviewService, private http: HttpClient, private feeder11: Feeder11Service) { }

  ngOnInit(): void {

    this.initMap();

  }

  initMap(): void {

    this.map = L.map('map').setView([22.7196, 75.8577], 7);

    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {

      attribution: '© OpenStreetMap contributors'

    }).addTo(this.map);

  }

  clearMarkers(): void {

    this.markers.forEach(marker => this.map?.removeLayer(marker));

    this.markers = [];

  }

  onRegionChange(): void {

    this.resetBelow('circle');

    this.mapview.getCircles(this.selectedRegion).subscribe((data: Circle[]) => {

      this.circles = data.filter(circle => circle.regionCode === this.selectedRegion);

      console.log('Filtered Circles:', this.circles);

    });

  }

  onCircleChange(): void {

    this.resetBelow('division');

    this.mapview.getDivisionsByCircle(this.selectedCircle).subscribe(data => {

      this.divisions = data.filter(division => division.circleCode === this.selectedCircle);

      console.log('Filtered Divisions:', this.divisions);

    });

  }

  onDivisionChange(): void {

    this.resetBelow('dc');

    this.mapview.getDC(this.selectedDivision).subscribe(data => {

      this.dcs = data

        .filter(dc => dc.divisionCode === this.selectedDivision)

        .map(dc => ({

          name: dc.distributionCenterName,

          code: dc.distributionCenterCode

        }));

      console.log('Selected Division Code:', this.selectedDivision);

      console.log('All DCs from API:', data);

      console.log('Filtered DC names:', this.dcs);

    });

  }

  resetBelow(level: 'circle' | 'division' | 'dc'): void {

    if (level === 'circle') {

      this.selectedCircle = '';

      this.selectedDivision = '';

      this.selectedDC = '';

      this.circles = [];

      this.divisions = [];

      this.dcs = [];

    } else if (level === 'division') {

      this.selectedDivision = '';

      this.selectedDC = '';

      this.divisions = [];

      this.dcs = [];

    } else if (level === 'dc') {

      this.selectedDC = '';

      this.dcs = [];

    }

  }

  getdashboard() {

    const body = {

      code\_of\_region: this.selectedRegion,

      code\_of\_circle: this.selectedCircle,

      code\_of\_division: this.selectedDivision,

      code\_of\_distribution\_center: this.selectedDC

    };

    this.http.post<any>('http://localhost:8080/getdashboardsearch', body).subscribe(data => {

      this.summary = data;

      console.log('Dashboard Summary:', data);

    });

    this.feeder11.getFeederCountDashboard(body).subscribe(countData => {

      this.feederCounts = countData;

      console.log('Feeder Count Dashboard:', countData);

    });

  }

  // viewMap(feeder: any): void {

  //   const body = {

  //     code\_of\_region: this.selectedRegion,

  //     code\_of\_circle: this.selectedCircle,

  //     code\_of\_division: this.selectedDivision,

  //     code\_of\_distribution\_center: this.selectedDC,

  //     code\_of\_feeder: feeder.kv\_11\_feeder\_code,

  //     type\_of\_feeder: 1

  //   };

  // this.feeder11.getRapdrpDTRMapview(body).subscribe(response => {

  //   const poleGeoJsonStr = response[0]?.get\_pole\_11\_kv;

  //   if (poleGeoJsonStr) {

  //     const poleGeoJson = JSON.parse(poleGeoJsonStr);

  //     const features = poleGeoJson.features || [];

  //     this.clearMarkers();

  // const pointMap = new Map<string, any>();

  // features.forEach(feature => {

  //   const poleCode = feature.properties.pole\_code;

  //   pointMap.set(poleCode, feature);

  //   console.log(poleCode);

  //   // console.log(features);

  // });

  // features.forEach((feature: any) => {

  //   const [lng, lat] = feature.geometry.coordinates;

  //   const marker = L.marker([lat, lng]).addTo(this.map!)

  //     .bindPopup(`<b>Pole Code:</b> ${feature.properties.pole\_code}`);

  //   this.markers.push(marker);

  //   const parentCode = feature.properties.parent\_pole\_code;

  //   const parentFeature = pointMap.get(parentCode);

  //   if (parentFeature) {

  //     const [parentLng, parentLat] = parentFeature.geometry.coordinates;

  //     L.polyline([[parentLat, parentLng], [lat, lng]], { color: 'green' }).addTo(this.map!);

  //   }

  // });

  // if (features.length > 0) {

  //   const [lng, lat] = features[0].geometry.coordinates;

  //   this.map?.setView([lat, lng], 15);

  // }

  viewMap(feeder: any): void {

    const body = {

      code\_of\_region: this.selectedRegion,

      code\_of\_circle: this.selectedCircle,

      code\_of\_division: this.selectedDivision,

      code\_of\_distribution\_center: this.selectedDC,

      code\_of\_feeder: feeder.kv\_11\_feeder\_code,

      type\_of\_feeder: 1

    };

    // Use forkJoin to call both APIs in parallel

    forkJoin({

      poleData: this.feeder11.getRapdrpPole11kvMapview(body),

      dtrData: this.feeder11.getRapdrpDTRMapview(body)

    }).subscribe(({ poleData, dtrData }) => {

      this.clearMarkers();

      console.log('DTR API Response:', dtrData);

      // Handle Pole Data

      let dtrFeatures: any[] = [];  // Declare dtrFeatures so it's accessible in the whole block

      const poleGeoJsonStr = poleData[0]?.get\_pole\_11\_kv;

      let poleFeatures: any[] = [];

      if (poleGeoJsonStr) {

        try {

          const parsed = typeof poleGeoJsonStr === 'string' ? JSON.parse(poleGeoJsonStr) : poleGeoJsonStr;

          poleFeatures = parsed.features || [];

        } catch (e) {

          console.error('Invalid Pole GeoJSON:', e);

        }

      }

      //     const poleMap = new Map<string, any>();

      //     poleFeatures.forEach(feature => {

      //       const poleCode = feature.properties.pole\_code;

      //       const parentPole=feature.properties.parent\_pole\_code;

      //       poleMap.set(poleCode, feature);

      //       console.log(feature);

      //       const [lng, lat] = feature.geometry.coordinates;

      //       const marker = L.marker([lat, lng], { icon: this.poleIcon() })

      //         .bindPopup(`<b>Pole Code:</b> ${poleCode} <br>

      //           <b>Parent Pole:</b> ${parentPole}`)

      //         .addTo(this.map!);

      //       this.markers.push(marker);

      //       const parentCode = feature.properties.parent\_pole\_code;

      //       const parentFeature = poleMap.get(parentCode);

      //       if (parentFeature) {

      //         const [parentLng, parentLat] = parentFeature.geometry.coordinates;

      //         L.polyline([[parentLat, parentLng], [lat, lng]], { color: 'green' }).addTo(this.map!);

      //       }

      //       else {

      //   // Try linking to DTR if parent is not a pole

      //   const parentDTR = dtrFeatures.find(dtr => dtr.properties?.pole\_code === parentCode);

      //   if (parentDTR && parentDTR.geometry?.coordinates) {

      //     const [dtrLng, dtrLat] = parentDTR.geometry.coordinates;

      //     L.polyline([[dtrLat, dtrLng], [lat, lng]], { color: 'blue' }).addTo(this.map!);

      //   }

      // }

      //     });

      const poleMap = new Map<string, any>();

      // Pass 1: Add markers and populate poleMap

      poleFeatures.forEach(feature => {

        const poleCode = feature.properties.pole\_code;

        const [lng, lat] = feature.geometry.coordinates;

        poleMap.set(poleCode, feature);

        const marker = L.marker([lat, lng], { icon: this.poleIcon() })

          .bindPopup(`<b>Pole Code:</b> ${poleCode} <br>

                <b>Parent Pole:</b> ${feature.properties.parent\_pole\_code}`)

          .addTo(this.map!);

        this.markers.push(marker);

      });

      // Pass 2: Draw lines from pole to its parent

      poleFeatures.forEach(feature => {

        // const poleCode = feature.properties.pole\_code;

        const parentCode = feature.properties.parent\_pole\_code;

        const [lng, lat] = feature.geometry.coordinates;

        const parentFeature = poleMap.get(parentCode);

        if (parentFeature) {

          const [parentLng, parentLat] = parentFeature.geometry.coordinates;

          L.polyline([[parentLat, parentLng], [lat, lng]], { color: 'green' }).addTo(this.map!);

        } else {

          // Try linking to DTR if parent is not a pole

          const parentDTR = dtrFeatures.find(dtr => dtr.properties?.pole\_code === parentCode);

          if (parentDTR && parentDTR.geometry?.coordinates) {

            const [dtrLng, dtrLat] = parentDTR.geometry.coordinates;

            L.polyline([[dtrLat, dtrLng], [lat, lng]], { color: 'blue' }).addTo(this.map!);

          }

          else{

            console.warn(`Parent not found for pole with parent code:${parentCode}`)

          }

        }

      });

      // Handle DTR Data (optional enhancement)

      const dtrJsonStr = dtrData[0]?.get\_dtr\_11\_04\_kv;

      console.log('Raw DTR GeoJSON String:', dtrJsonStr);

      if (dtrJsonStr) {

        try {

          const parsedDTR = JSON.parse(dtrJsonStr);

          dtrFeatures = parsedDTR.features || [];

          dtrFeatures.forEach(dtr => {

            if (dtr.geometry?.coordinates) {

              const [lng, lat] = dtr.geometry.coordinates;

              const marker = L.marker([lat, lng], { icon: this.dtrIcon() })

                .bindPopup(`<b>Pole Code:</b> ${dtr.properties?.pole\_code || 'N/A'} <br>

                <b>DTR Code:</b> ${dtr.properties?.dtr\_code || 'N/A'} <br>

                <b>DTR Name:</b> ${dtr.properties?.dtr\_name || 'N/A'} <br>

                <b>DTR Capacity:</b> ${dtr.properties?.dtr\_capacity || 'N/A'} <br>

                <b>Parent Pole</b> ${dtr.properties?.parent\_pole\_code || 'N/A'}`)

                .addTo(this.map!);

              this.markers.push(marker);

            }

            else {

              console.warn('Invalid DTR geometry:', dtr);

            }

          });

        } catch (e) {

          console.error('Invalid DTR GeoJSON:', e);

        }

      }

      else {

        console.warn('No DTR GeoJSON string found in response.');

      }

      // Zoom to first pole

      if (poleFeatures.length > 0) {

        const [lng, lat] = poleFeatures[0].geometry.coordinates;

        this.map?.setView([lat, lng], 15);

      }

    }, error => {

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

    });

  }

  poleIcon(): L.Icon {

    return L.icon({

      iconUrl: 'assets/pole-icon.png',

      iconSize: [25, 41],

      iconAnchor: [12, 41]

    });

  }

  dtrIcon(): L.Icon {

    return L.icon({

      iconUrl: 'assets/dtr-icon.png',

      iconSize: [25, 41],

      iconAnchor: [12, 41]

    });

  }

}