/\*\*

\* UK Anaerobic Digestion Mapping Application

\* Complete MapLibre GL JS application with interactive features

\*/

console.log('🚀 Script file loading...');

// ==================== CONSTANTS AND CONFIGURATION ====================

const URLS = {

AD\_CSV: "https://storage.googleapis.com/moaads\_maps/ad\_plants3.csv",

MANURE\_LAD\_GROUPS: "https://storage.googleapis.com/moaads\_maps/manure\_by\_LAD\_groups.csv",

MANURE\_LPA\_GROUPS: "https://storage.googleapis.com/moaads\_maps/manure\_by\_LPA\_groups.csv",

LAD\_TOPO: "https://storage.googleapis.com/moaads\_maps/LADS.topojson",

LPA\_TOPO: "https://storage.googleapis.com/moaads\_maps/LPA.topojson",

AONB\_TOPO: "https://storage.googleapis.com/moaads\_maps/AONB.topojson",

SSSI\_TOPO: "https://storage.googleapis.com/moaads\_maps/SSSI.topojson",

NVZ\_TOPO: "https://storage.googleapis.com/moaads\_maps/NVZ2021.topojson",

FLOOD\_TOPO: "https://storage.googleapis.com/moaads\_maps/Historic\_Flood\_Map.topojson",

ALC\_TOPO: "https://storage.googleapis.com/moaads\_maps/ALC.topojson",

DNO\_TOPO: "https://storage.googleapis.com/moaads\_maps/DNO\_fixed.topojson",

WATER\_TOPO: "https://storage.googleapis.com/moaads\_maps/water\_companies.topojson",

POP\_TOPO: "https://storage.googleapis.com/moaads\_maps/LADSPopulation2021.topojson",

NTS\_KML: "https://storage.googleapis.com/moaads\_maps/NTS.kml",

ROADS\_MB: "https://storage.googleapis.com/moaads\_maps/oproad\_gb.mbtiles",

BROWNFIELD\_TOPO: "https://storage.googleapis.com/moaads\_maps/brownfield-land.topojson",

MANURE\_VOLUMES\_TOPO: "https://storage.googleapis.com/moaads\_maps/manure\_by\_LAD\_groups.csv"

};

const MAP\_CONFIG = {

INITIAL\_CENTER: [-2.5, 54.5],

INITIAL\_ZOOM: 6,

MAX\_ZOOM: 18,

MIN\_ZOOM: 5

};

const AD\_GROUP\_COLORS = {

'Operational': '#0f5132',

'Operational: GtG + GtG&CHP': '#0f5132',

'Operational: CHP': '#198754',

'Operational: Other': '#6f9c6f',

'Under Construction': '#2574ce',

'Planning Granted': '#854ec2',

'Planning Application': '#f38c1d',

'Planning Submitted': '#f38c1d',

'Proposed': '#f38c1d',

'Refused': '#b6222b',

'Withdrawn': '#b6222b',

'Other': '#6b7280'

};

// ========== MANURE VOLUMES LAYER ==========

let manureVolumesLayer = null;

let manureVolumesVisibility = {

'Beef FYM': false,

'Beef Slurry': false,

'Dairy FYM': false,

'Dairy Slurry': false,

'Broilers': false,

'Layers': false,

'Pigs': false,

'Sheep': false

};

const manureVolumesColors = {

'Beef FYM': '#8B4513', // Dark brown

'Beef Slurry': '#A0522D', // Lighter brown

'Dairy FYM': '#4682B4', // Steel blue

'Dairy Slurry': '#5F9EA0', // Cadet blue

'Broilers': '#FF6347', // Tomato

'Layers': '#FF7F50', // Coral

'Pigs': '#DA70D6', // Orchid

'Sheep': '#32CD32' // Lime green

};

// Field mappings for each livestock category

const manureFieldMappings = {

'Beef FYM': ['B\_FYM\_Gras', 'B\_FYM\_AraW', 'B\_FYM\_AraS', 'B\_Grazing', 'Beef\_FYM', 'BEEF\_FYM', 'beef\_fym', 'Beef FYM\_tonnes'],

'Beef Slurry': ['B\_Slu\_Gras', 'B\_Slu\_AraS', 'B\_Slu\_AraW', 'Beef\_Slurry', 'BEEF\_SLURRY', 'beef\_slurry', 'Beef Slurry\_tonnes'],

'Dairy FYM': ['D\_FYM\_Gras', 'D\_FYM\_AraW', 'D\_FYM\_AraS', 'D\_Grazing', 'Dairy\_FYM', 'DAIRY\_FYM', 'dairy\_fym', 'Dairy FYM\_tonnes'],

'Dairy Slurry': ['D\_Slu\_Gras', 'D\_Slu\_AraS', 'D\_Slu\_AraW', 'Dairy\_Slurry', 'DAIRY\_SLURRY', 'dairy\_slurry', 'Dairy Slurry\_tonnes'],

'Broilers': ['Bro\_FYM\_Gr', 'Bro\_FYM\_Ar', 'Bro\_FYM\_\_1', 'Broilers', 'BROILERS', 'broilers', 'Broiler', 'BROILER', 'Broilers\_tonnes'],

'Layers': ['L\_FYM\_Gras', 'L\_FYM\_AraW', 'L\_FYM\_AraS', 'L\_FrRngGra', 'Layers', 'LAYERS', 'layers', 'Layer', 'LAYER', 'Layers\_tonnes'],

'Pigs': ['P\_FYM\_Gras', 'P\_FYM\_AraW', 'P\_FYM\_AraS', 'P\_Slu\_Gras', 'P\_Slu\_AraS', 'P\_Slu\_AraW', 'P\_OutGrazi', 'Pigs', 'PIGS', 'pigs', 'Pig', 'PIG', 'Pigs\_tonnes'],

'Sheep': ['S\_FYM\_Gras', 'S\_FYM\_AraW', 'S\_FYM\_AraS', 'S\_Grazing', 'Sheep', 'SHEEP', 'sheep', 'Sheep\_tonnes']

};

// Manure Volumes Layer Configuration - 8 livestock categories from CSV data

const MANURE\_VOLUMES\_URL = "https://storage.googleapis.com/moaads\_maps/manure\_volumes\_wgs84.topojson";

// Store data values for dynamic color calculation

let manureDataValues = [];

const MANURE\_LAYER\_CONFIGS = [

{ id: 'beef-fym', name: 'Beef FYM', color: '#8B4513', field: 'Beef FYM\_tonnes' },

{ id: 'beef-slurry', name: 'Beef Slurry', color: '#A0522D', field: 'Beef Slurry\_tonnes' },

{ id: 'dairy-fym', name: 'Dairy FYM', color: '#4682B4', field: 'Dairy FYM\_tonnes' },

{ id: 'dairy-slurry', name: 'Dairy Slurry', color: '#5F9EA0', field: 'Dairy Slurry\_tonnes' },

{ id: 'broilers', name: 'Broilers', color: '#FF6347', field: 'Broilers\_tonnes' },

{ id: 'layers', name: 'Layers', color: '#FF7F50', field: 'Layers\_tonnes' },

{ id: 'pigs', name: 'Pigs', color: '#DA70D6', field: 'Pigs\_tonnes' },

{ id: 'sheep', name: 'Sheep', color: '#32CD32', field: 'Sheep\_tonnes' }

];

const LAYER\_GROUPS = [

{

id: 'manure',

name: 'Manure Volumes by Livestock',

icon: '📊',

layers: MANURE\_LAYER\_CONFIGS.map(config => config.id),

defaultOpen: false

},

{

id: 'boundaries',

name: 'Boundaries',

icon: '🗺️',

layers: ['lad', 'lpa']

},

{

id: 'environmental',

name: 'Environmental',

icon: '🍃',

layers: ['aonb', 'sssi', 'nvz', 'flood']

},

{

id: 'agricultural',

name: 'Agricultural Land Classification',

icon: '🌾',

layers: ['alc-grade-1', 'alc-grade-2', 'alc-grade-3', 'alc-grade-4', 'alc-grade-5']

},

{

id: 'roads',

name: 'Roads',

icon: '🛣️',

layers: ['roads']

},

{

id: 'infrastructure',

name: 'Infrastructure',

icon: '⚡',

layers: ['dno', 'water', 'nts', 'brownfield']

}

];

// ==================== GLOBAL STATE ====================

let map = null;

let proximityAnalysisMode = false;

let proximityMarker = null;

let layerManager = null;

let adPlantsData = [];

let manureData = [];

let ladBoundariesData = null;

let lpaData = null;

let ladManureData = [];

let lpaManureData = [];

let searchFuse = null;

let activeLayers = new Set(['lad']);

let currentInfoPanelData = null;

let adLayersLoaded = false;

let manureLayersLoaded = false;

let environmentalLayersLoaded = false;

let infrastructureLayersLoaded = false;

let agriculturalLayersLoaded = false;

let manureVolumesData = null;

let manureVolumesCSVData = null;

// Event handler registry for cleanup management

const layerEventHandlers = new Map();

// Data validation constants

const TOPOJSON\_SCHEMA = {

requiredFields: ['type', 'objects'],

geometryTypes: ['Point', 'LineString', 'Polygon', 'MultiPoint', 'MultiLineString', 'MultiPolygon']

};

const GEOJSON\_SCHEMA = {

requiredFields: ['type', 'features'],

featureSchema: {

requiredFields: ['type', 'geometry', 'properties'],

geometryTypes: ['Point', 'LineString', 'Polygon', 'MultiPoint', 'MultiLineString', 'MultiPolygon']

}

};

// ==================== UTILITY FUNCTIONS ====================

// Data validation functions

function validateTopoJSON(data, layerName = 'Unknown') {

if (!data || typeof data !== 'object') {

throw new Error(`Invalid TopoJSON for ${layerName}: Data is not an object`);

}

// Check required fields

for (const field of TOPOJSON\_SCHEMA.requiredFields) {

if (!data.hasOwnProperty(field)) {

throw new Error(`Invalid TopoJSON for ${layerName}: Missing required field '${field}'`);

}

}

// Validate objects structure

if (!data.objects || typeof data.objects !== 'object') {

throw new Error(`Invalid TopoJSON for ${layerName}: 'objects' field must be an object`);

}

const objectKeys = Object.keys(data.objects);

if (objectKeys.length === 0) {

throw new Error(`Invalid TopoJSON for ${layerName}: No objects found in TopoJSON`);

}

console.log(`✅ TopoJSON validation passed for ${layerName}:`, {

type: data.type,

objectKeys: objectKeys,

arcsCount: data.arcs ? data.arcs.length : 0

});

return true;

}

function validateGeoJSON(data, layerName = 'Unknown') {

if (!data || typeof data !== 'object') {

throw new Error(`Invalid GeoJSON for ${layerName}: Data is not an object`);

}

// Check required fields

for (const field of GEOJSON\_SCHEMA.requiredFields) {

if (!data.hasOwnProperty(field)) {

throw new Error(`Invalid GeoJSON for ${layerName}: Missing required field '${field}'`);

}

}

// Validate features array

if (!Array.isArray(data.features)) {

throw new Error(`Invalid GeoJSON for ${layerName}: 'features' must be an array`);

}

if (data.features.length === 0) {

console.warn(`⚠️ Warning: GeoJSON for ${layerName} contains no features`);

return true;

}

// Validate first few features for structure

const sampleSize = Math.min(3, data.features.length);

for (let i = 0; i < sampleSize; i++) {

const feature = data.features[i];

for (const field of GEOJSON\_SCHEMA.featureSchema.requiredFields) {

if (!feature.hasOwnProperty(field)) {

throw new Error(`Invalid GeoJSON for ${layerName}: Feature ${i} missing required field '${field}'`);

}

}

// Validate geometry type

if (feature.geometry && feature.geometry.type) {

if (!GEOJSON\_SCHEMA.featureSchema.geometryTypes.includes(feature.geometry.type)) {

throw new Error(`Invalid GeoJSON for ${layerName}: Feature ${i} has unsupported geometry type '${feature.geometry.type}'`);

}

}

}

console.log(`✅ GeoJSON validation passed for ${layerName}:`, {

type: data.type,

featureCount: data.features.length,

geometryTypes: [...new Set(data.features.map(f => f.geometry?.type).filter(Boolean))]

});

return true;

}

// Event handler management

function registerLayerEventHandlers(layerId, handlers) {

if (!layerEventHandlers.has(layerId)) {

layerEventHandlers.set(layerId, []);

}

const handlerList = layerEventHandlers.get(layerId);

handlers.forEach(handler => {

handlerList.push(handler);

map.on(handler.event, layerId, handler.callback);

});

console.log(`Registered ${handlers.length} event handlers for layer ${layerId}`);

}

function cleanupLayerEventHandlers(layerId) {

if (layerEventHandlers.has(layerId)) {

const handlers = layerEventHandlers.get(layerId);

handlers.forEach(handler => {

try {

map.off(handler.event, layerId, handler.callback);

} catch (error) {

console.warn(`Failed to remove event handler for ${layerId}:`, error);

}

});

layerEventHandlers.delete(layerId);

console.log(`Cleaned up ${handlers.length} event handlers for layer ${layerId}`);

}

}

function cleanupAllEventHandlers() {

const layerIds = Array.from(layerEventHandlers.keys());

layerIds.forEach(layerId => cleanupLayerEventHandlers(layerId));

console.log(`Cleaned up all event handlers for ${layerIds.length} layers`);

}

// Promise timeout wrapper for better error handling

async function withTimeout(promise, timeoutMs = 30000, operationName = 'Operation') {

const timeoutPromise = new Promise((\_, reject) => {

setTimeout(() => {

reject(new Error(`${operationName} timed out after ${timeoutMs}ms`));

}, timeoutMs);

});

return Promise.race([promise, timeoutPromise]);

}

function showToast(message, type = 'info') {

const container = document.getElementById('toast-container');

const toast = document.createElement('div');

const bgColor = type === 'error' ? 'bg-red-500' : type === 'success' ? 'bg-green-500' : 'bg-blue-500';

toast.className = `${bgColor} text-white px-4 py-2 rounded-lg shadow-lg transform translate-y-2 opacity-0 transition-all duration-300`;

toast.textContent = message;

container.appendChild(toast);

// Animate in

setTimeout(() => {

toast.classList.remove('translate-y-2', 'opacity-0');

}, 100);

// Remove after 3 seconds

setTimeout(() => {

toast.classList.add('translate-y-2', 'opacity-0');

setTimeout(() => container.removeChild(toast), 300);

}, 3000);

}

function hideLoading() {

const overlay = document.getElementById('loading-overlay');

if (overlay) {

overlay.style.display = 'none';

}

}

function createMapStyle() {

return {

version: 8,

sources: {

'osm': {

type: 'raster',

tiles: ['https://tile.openstreetmap.org/{z}/{x}/{y}.png'],

tileSize: 256,

attribution: '© OpenStreetMap contributors'

}

},

layers: [

{

id: 'osm',

type: 'raster',

source: 'osm'

}

]

};

}

// ==================== LAD CLICK HANDLER ====================

async function handleLADClick(e) {

const properties = e.features[0].properties;

const ladCode = properties.LAD23CD;

const ladName = properties.LAD23NM;

console.log('LAD polygon clicked:', ladName);

// 1. Immediate Visual Feedback - Change polygon styling

try {

map.setPaintProperty('lad-fill', 'fill-opacity', 0.15);

map.setPaintProperty('lad', 'line-width', 2);

} catch (error) {

console.warn('Failed to update LAD visual feedback:', error);

}

// 2. Show Loading Panel

showLADLoadingPanel(ladName);

try {

// 3. Load required data

await Promise.all([

loadADPlantsDataIfNeeded(),

loadManureDataIfNeeded()

]);

// 4. Perform spatial analysis to find AD plants in LAD

const plantsInLAD = await findPlantsInLAD(ladCode, properties);

// 5. Get manure data for LAD

const associatedManure = manureData.find(row => row['LAD code'] === ladCode);

// 6. Display comprehensive information panel

openInfoPanel({

type: 'lad',

ladProperties: properties,

plantsInLAD,

manureData: associatedManure

});

} catch (error) {

console.error('Error processing LAD click:', error);

showLADErrorPanel(ladName, error.message);

}

}

function showLADLoadingPanel(ladName) {

const panel = document.getElementById('info-panel');

const title = document.getElementById('info-panel-title');

const content = document.getElementById('info-panel-content');

if (!panel) return;

if (title) title.textContent = `📍 ${ladName}`;

if (content) {

content.innerHTML = `

<div class="loading-content text-center py-8">

<div class="loading-spinner w-8 h-8 border-2 border-blue-600 border-t-transparent rounded-full mx-auto mb-4"></div>

<p class="text-slate-600">Loading LAD data...</p>

</div>

`;

}

panel.classList.remove('translate-x-full');

panel.classList.add('translate-x-0');

}

async function loadADPlantsDataIfNeeded() {

if (adPlantsData.length === 0) {

console.log('Loading AD plants data for LAD analysis...');

adPlantsData = await loadCSVData(URLS.AD\_CSV, 'AD Plants data');

console.log('AD plants data loaded:', adPlantsData.length);

}

}

async function loadManureDataIfNeeded() {

if (manureData.length === 0) {

console.log('Loading manure data for LAD analysis...');

manureData = await loadCSVData(URLS.MANURE\_LAD\_GROUPS, 'Manure by LAD data');

console.log('Manure data loaded:', manureData.length);

}

}

async function loadManureVolumesDataIfNeeded() {

if (manureVolumesData && manureVolumesData.features) return;

try {

console.log('Loading manure volumes data for proximity analysis...');

if (!manureLayersLoaded) {

await loadManureLayers();

}

console.log('Manure volumes data loaded for proximity analysis');

} catch (error) {

console.error('Failed to load manure volumes data:', error);

throw error;

}

}

async function findPlantsInLAD(ladCode, ladProperties) {

// Use exact field definitions - no guessing

const plantsInLAD = adPlantsData.filter(plant => {

return plant.lad\_code === ladCode;

});

console.log(`Found ${plantsInLAD.length} plants in LAD ${ladProperties.LAD23NM} using code ${ladCode}`);

if (plantsInLAD.length > 0) {

console.log('Sample plant data:', plantsInLAD[0]);

}

return plantsInLAD;

}

// LAD Info Panel Display - using LPA style formatting

function displayLADInfoPanel({ ladProperties, plantsInLAD, manureData }) {

console.log('Displaying LAD info panel');

const panel = document.getElementById('info-panel');

const title = document.getElementById('info-panel-title');

const content = document.getElementById('info-panel-content');

if (!panel || !title || !content) {

console.error('Info panel elements not found');

return;

}

const ladName = ladProperties.LAD23NM || 'Unknown LAD';

const ladCode = ladProperties.LAD23CD;

console.log(`Setting up info panel for LAD: ${ladName}`);

title.textContent = `📍 ${ladName}`;

// Group plants by group (which matches the AD layer groups)

const statusGroups = {};

plantsInLAD.forEach(plant => {

const group = plant.Group || 'Unknown';

if (!statusGroups[group]) {

statusGroups[group] = [];

}

statusGroups[group].push(plant);

});

// Group colors matching AD plant layer colors

const statusColors = {

'Operational: GtG + GtG&CHP': '#0f5132',

'Operational: CHP': '#198754',

'Operational: Other': '#6f9c6f',

'Planning Submitted': '#f38c1d',

'Planning Refused / Withdrawn / Abandoned': '#6b7280',

'Planning Granted': '#854ec2',

'Under Construction': '#2574ce',

'Operational': '#0f5132',

'Planning Application': '#f38c1d',

'Proposed': '#f38c1d',

'Refused': '#b6222b',

'Withdrawn': '#b6222b'

};

// Shortened status names for display

const statusDisplayNames = {

'Planning Refused / Withdrawn / Abandoned': 'Refused/Withdrawn/Abandoned'

};

// Calculate statistics

const totalPlants = plantsInLAD.length;

const operationalPlants = plantsInLAD.filter(p => p.Status && p.Status.includes('Operational')).length;

const totalCapacity = plantsInLAD.reduce((sum, plant) => {

const capacity = parseFloat(String(plant['Capacity(kWe)']).replace(/,/g, '')) || 0;

return sum + capacity;

}, 0);

// Calculate total GWh capacity for overview

const totalGWhCapacity = plantsInLAD.reduce((sum, plant) => {

const gwh = parseFloat(plant.GWh) || 0;

return sum + gwh;

}, 0);

let html = `

<div class="space-y-6">

<!-- Header Card -->

<div class="bg-gradient-to-br from-blue-600 to-indigo-700 text-white border border-blue-800 p-6">

<div class="mb-4">

<h2 class="text-xl font-bold">${ladName}</h2>

</div>

<!-- Key Metrics Grid -->

<div class="grid grid-cols-2 gap-3">

<div class="bg-white bg-opacity-10 border border-white border-opacity-20 p-3 text-center h-16 flex flex-col justify-center">

<div class="text-lg font-bold">${totalPlants}</div>

<div class="text-xs opacity-90">Total Plants</div>

</div>

<div class="bg-white bg-opacity-10 border border-white border-opacity-20 p-3 text-center h-16 flex flex-col justify-center">

<div class="text-lg font-bold text-green-300">${operationalPlants}</div>

<div class="text-xs opacity-90">Operational</div>

</div>

<div class="bg-white bg-opacity-10 border border-white border-opacity-20 p-3 text-center h-16 flex flex-col justify-center">

<div class="text-lg font-bold text-yellow-300">${formatNumber(Math.round(totalCapacity / 1000))}</div>

<div class="text-xs opacity-90">MWe Capacity</div>

</div>

<div class="bg-white bg-opacity-10 border border-white border-opacity-20 p-3 text-center h-16 flex flex-col justify-center">

<div class="text-lg font-bold text-orange-300">${formatNumber(totalGWhCapacity)}</div>

<div class="text-xs opacity-90">GWh/year</div>

</div>

</div>

</div>

`;

// Plants by Status - Modern Card Design

if (Object.keys(statusGroups).length > 0) {

html += `

<div class="bg-white border border-gray-200 overflow-hidden">

<div class="bg-gray-50 px-6 py-4 border-b border-gray-200">

<button class="w-full text-left flex items-center justify-between font-semibold text-gray-800 hover:text-blue-600 transition-colors" onclick="togglePlantsContent(this)">

<div class="flex items-center">

<div class="w-10 h-10 bg-blue-600 rounded-lg flex items-center justify-center mr-3">

<svg class="w-5 h-5 text-white" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 21V5a2 2 0 00-2-2H7a2 2 0 00-2 2v16m14 0h2m-2 0h-5m-9 0H3m2 0h5M9 7h1m-1 4h1m4-4h1m-1 4h1m-5 10v-5a1 1 0 011-1h2a1 1 0 011 1v5m-4 0h4"></path>

</svg>

</div>

<div>

<h3 class="text-lg font-semibold">AD Plants by Status</h3>

<p class="text-sm text-gray-600">${totalPlants} plants in this area</p>

</div>

</div>

<svg class="w-5 h-5 text-gray-400 chevron-icon" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 9l-7 7-7-7"></path>

</svg>

</button>

</div>

<div class="plants-content" style="display: none;">

`;

Object.entries(statusGroups).forEach(([status, plants]) => {

const color = statusColors[status] || '#6b7280';

const displayName = statusDisplayNames[status] || status;

html += `

<div class="border-b border-gray-100 last:border-b-0">

<button class="w-full text-left px-6 py-3 text-white font-medium flex items-center justify-between transition-all duration-200 hover:opacity-90" style="background: linear-gradient(135deg, ${color} 0%, ${color}dd 100%);" onclick="toggleNextElement(this)">

<span>${displayName}</span>

<div class="flex items-center">

<span class="bg-white bg-opacity-20 px-2 py-1 text-sm mr-2">${plants.length}</span>

<svg class="w-4 h-4 chevron-icon" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 9l-7 7-7-7"></path>

</svg>

</div>

</button>

<div style="display: none;" class="px-6 py-4 bg-gray-50">

<div class="space-y-4">

${plants.map(plant => {

const capacity = parseFloat(String(plant['Capacity(kWe)']).replace(/,/g, '')) || 0;

const totalFeedstock = plant['Totalfeedstock(tpa)'] || plant.Totalfeedstock || 'N/A';

const gwh = parseFloat(plant.GWh) || 0;

return `

<div class="bg-white border border-gray-200 hover:border-blue-300 hover:shadow-md cursor-pointer transition-all duration-200 p-4" onclick="flyToPlant(${plant.Latitude}, ${plant.Longitude}, '${(plant['Site name'] || plant['Site Name'] || 'Unknown Site').replace(/'/g, "\\'")}')">

<div class="flex items-start justify-between mb-3">

<div class="flex-1">

<h4 class="font-bold text-gray-900 hover:text-blue-600 transition-colors text-lg mb-1">${plant['Site name'] || plant['Site Name'] || 'Unknown Site'}</h4>

<div class="text-sm text-gray-600 mb-2">${plant.Developer || 'Unknown Developer'}</div>

</div>

<div class="text-right">

<div class="text-lg font-bold text-blue-600">${formatNumber(capacity)} kWe</div>

${gwh > 0 ? `<div class="text-sm text-green-600">${formatNumber(gwh)} GWh/yr</div>` : ''}

</div>

</div>

<div class="grid grid-cols-2 gap-3 text-sm">

<div class="space-y-2">

<div class="flex items-center text-gray-600">

<svg class="w-4 h-4 mr-2 text-gray-400" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M17.657 16.657L13.414 20.9a1.998 1.998 0 01-2.827 0l-4.244-4.243a8 8 0 1111.314 0z"></path>

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M15 11a3 3 0 11-6 0 3 3 0 016 0z"></path>

</svg>

<span class="truncate">${plant.Address || plant.Postcode || 'Location N/A'}</span>

</div>

<div class="flex items-center text-gray-600">

<svg class="w-4 h-4 mr-2 text-gray-400" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M8 7V3m8 4V3m-9 8h10M5 21h14a2 2 0 002-2V7a2 2 0 00-2-2H5a2 2 0 00-2 2v12a2 2 0 002 2z"></path>

</svg>

<span>${plant.Completion || 'Date N/A'}</span>

</div>

</div>

<div class="space-y-2">

<div class="flex items-center text-gray-600">

<svg class="w-4 h-4 mr-2 text-gray-400" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M20 7l-8-4-8 4m16 0l-8 4m8-4v10l-8 4m0-10L4 7m8 4v10M4 7v10l8 4"></path>

</svg>

<span class="font-medium text-orange-600">${totalFeedstock} tpa</span>

</div>

<div class="flex items-center text-gray-600">

<svg class="w-4 h-4 mr-2 text-gray-400" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19.428 15.428a2 2 0 00-1.022-.547l-2.387-.477a6 6 0 00-3.86.517l-.318.158a6 6 0 01-3.86.517L6.05 15.21a2 2 0 00-1.806.547M8 4h8l-1 1v5.172a2 2 0 00.586 1.414l5 5c1.26 1.26.367 3.414-1.415 3.414H4.828c-1.782 0-2.674-2.154-1.414-3.414l5-5A2 2 0 009 10.172V5L8 4z"></path>

</svg>

<span class="text-xs">${plant.Type || 'Type N/A'}</span>

</div>

</div>

</div>

<div class="mt-3 pt-3 border-t border-gray-200">

<div class="text-xs text-gray-500 truncate">

<strong>Feedstock:</strong> ${plant.Feedstock || 'Not specified'}

</div>

</div>

</div>

`;

}).join('')}

</div>

</div>

</div>

`;

});

html += `

</div>

</div>

`;

} else {

html += `

<div class="bg-white border border-gray-200 p-8 text-center">

<div class="w-16 h-16 bg-gray-100 flex items-center justify-center mx-auto mb-4">

<svg class="w-8 h-8 text-gray-400" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 21V5a2 2 0 00-2-2H7a2 2 0 00-2 2v16m14 0h2m-2 0h-5m-9 0H3m2 0h5M9 7h1m-1 4h1m4-4h1m-1 4h1m-5 10v-5a1 1 0 011-1h2a1 1 0 011 1v5m-4 0h4"></path>

</svg>

</div>

<h3 class="text-lg font-semibold text-gray-900 mb-2">No AD Plants Found</h3>

<p class="text-gray-600">There are currently no anaerobic digestion plants in this LAD area.</p>

</div>

`;

}

// Manure Analysis - starting collapsed

console.log('Looking for manure data for LAD:', ladName, ladCode);

console.log('Available manure data entries:', window.ladManureData ? window.ladManureData.length : 'NO DATA');

const ladManureEntry = findManureDataForLAD(ladName, ladCode);

console.log('Found manure entry:', ladManureEntry);

if (ladManureEntry) {

// Handle both string and number values for totals

const totalTonnesValue = ladManureEntry['total\_tonnes'];

const totalGwhValue = ladManureEntry['total\_gwh'];

const totalTonnes = parseFloat(

typeof totalTonnesValue === 'string' ? totalTonnesValue.replace(/,/g, '') : totalTonnesValue || 0

);

const totalGwh = parseFloat(

typeof totalGwhValue === 'string' ? totalGwhValue.replace(/,/g, '') : totalGwhValue || 0

);

html += `

<div class="bg-white border border-gray-200 overflow-hidden">

<div class="bg-gradient-to-r from-green-50 to-amber-50 px-6 py-4 border-b border-gray-200">

<button class="w-full text-left flex items-center justify-between font-semibold text-gray-800 hover:text-green-600 transition-colors" onclick="toggleManureContent(this)">

<div class="flex items-center">

<div class="w-10 h-10 bg-green-600 rounded-lg flex items-center justify-center mr-3">

<svg class="w-5 h-5 text-white" fill="currentColor" viewBox="0 0 24 24">

<path d="M12.5 8.5v7h-1v-7h1zm-8 7V12a4 4 0 014-4h8a4 4 0 014 4v3.5h-1V12a3 3 0 00-3-3h-8a3 3 0 00-3 3v3.5h-1zm12-11.5a1.5 1.5 0 11-3 0 1.5 1.5 0 013 0zm1 0a2.5 2.5 0 10-5 0 2.5 2.5 0 005 0zm-2.5-1.5a.5.5 0 00-.5.5v1a.5.5 0 001 0V3a.5.5 0 00-.5-.5z"/>

<path d="M12 20c-4.4 0-8-3.6-8-8s3.6-8 8-8 8 3.6 8 8-3.6 8-8 8zm0-1c3.9 0 7-3.1 7-7s-3.1-7-7-7-7 3.1-7 7 3.1 7 7 7z"/>

<path d="M12 6c-1.1 0-2 .9-2 2v4c0 1.1.9 2 2 2s2-.9 2-2V8c0-1.1-.9-2-2-2z"/>

</svg>

</div>

<div>

<h3 class="text-lg font-semibold">Manure Production Analysis</h3>

<p class="text-sm text-gray-600">Annual livestock waste generation</p>

</div>

</div>

<svg class="w-5 h-5 text-gray-400 chevron-icon" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 9l-7 7-7-7"></path>

</svg>

</button>

</div>

<div class="manure-content" style="display: none;">

<div class="p-6">

<!-- Summary Stats -->

<div class="grid grid-cols-2 gap-3 mb-6">

<div class="bg-gradient-to-br from-green-100 to-green-200 border border-green-300 p-2 text-center h-14 flex flex-col justify-center">

<div class="text-sm font-bold text-green-700">${formatNumber(Math.round(totalTonnes))}</div>

<div class="text-xs text-green-600 font-medium truncate">Tonnes/Year</div>

</div>

<div class="bg-gradient-to-br from-amber-100 to-orange-200 border border-orange-300 p-2 text-center h-14 flex flex-col justify-center">

<div class="text-sm font-bold text-orange-700">${formatNumber(Math.round(totalGwh))}</div>

<div class="text-xs text-orange-600 font-medium truncate">GWh Potential</div>

</div>

</div>

<!-- Manure Breakdown -->

<div class="space-y-3">

<h4 class="font-semibold text-gray-800 mb-3">Breakdown by Livestock Type</h4>

${generateManureBreakdown(ladManureEntry)}

</div>

</div>

</div>

</div>

`;

} else {

console.log('No manure data found for this LAD');

// Show that no data was found

if (window.ladManureData && window.ladManureData.length > 0) {

html += `

<div class="bg-white border border-gray-200 p-8 text-center">

<div class="w-16 h-16 bg-gray-100 flex items-center justify-center mx-auto mb-4">

<svg class="w-8 h-8 text-gray-400" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M9 12h6m-6 4h6m2 5H7a2 2 0 01-2-2V5a2 2 0 012-2h5.586a1 1 0 01.707.293l5.414 5.414a1 1 0 01.293.707V19a2 2 0 01-2 2z"></path>

</svg>

</div>

<h3 class="text-lg font-semibold text-gray-900 mb-2">No Manure Data</h3>

<p class="text-gray-600">Manure production data is not available for this LAD area.</p>

</div>

`;

}

}

// Add Spatial Analysis and Quick Actions only if we have plants

if (plantsInLAD.length > 0) {

const operationalPlants = plantsInLAD.filter(p => p.Status && p.Status.includes('Operational'));

const avgCapacity = totalCapacity / Math.max(totalPlants, 1);

const manureFeedstock = plantsInLAD.reduce((sum, plant) => {

return sum + (parseFloat(String(plant['Manure/Slurry(tpa)']).replace(/,/g, '')) || 0);

}, 0);

html += `

<div class="bg-white border rounded-lg">

<button class="w-full text-left px-4 py-3 font-semibold text-gray-800 hover:bg-gray-50" onclick="this.nextElementSibling.style.display = this.nextElementSibling.style.display === 'none' ? 'block' : 'none'">

📊 Spatial Analysis

</button>

<div style="display: none;" class="border-t p-4">

<div class="space-y-3 text-sm">

<div class="grid grid-cols-2 gap-4">

<div>

<div class="text-gray-600">Plant Density</div>

<div class="font-medium">${totalPlants} plants in LAD</div>

</div>

<div>

<div class="text-gray-600">Avg Capacity</div>

<div class="font-medium">${formatNumber(Math.round(avgCapacity))} kWe</div>

</div>

</div>

<div class="grid grid-cols-2 gap-4">

<div>

<div class="text-gray-600">Manure Utilization</div>

<div class="font-medium">${formatNumber(manureFeedstock)} tpa</div>

</div>

<div>

<div class="text-gray-600">Development Status</div>

<div class="font-medium">${Math.round((operationalPlants.length / Math.max(totalPlants, 1)) \* 100)}% operational</div>

</div>

</div>

</div>

</div>

</div>

`;

}

// Quick Actions Section (always shown)

html += `

<div class="bg-white border rounded-lg">

<button class="w-full text-left px-4 py-3 font-semibold text-gray-800 hover:bg-gray-50" onclick="this.nextElementSibling.style.display = this.nextElementSibling.style.display === 'none' ? 'block' : 'none'">

⚡ Quick Actions

</button>

<div style="display: none;" class="border-t p-3">

<div class="space-y-2">

<button onclick="exportLADData('${ladCode}', '${ladName.replace(/'/g, "\\'")}', ${totalPlants}, ${JSON.stringify(ladManureEntry || {}).replace(/"/g, '&quot;')})"

class="w-full bg-blue-600 text-white px-3 py-2 rounded text-sm hover:bg-blue-700 transition-colors">

📄 Export LAD Report

</button>

<button onclick="showLADOnMap('${ladCode}')"

class="w-full bg-green-600 text-white px-3 py-2 rounded text-sm hover:bg-green-700 transition-colors">

🗺️ Center on Map

</button>

<button onclick="compareLADToRegion('${ladCode}', '${ladName.replace(/'/g, "\\'")}', ${totalPlants}, ${totalCapacity})"

class="w-full bg-purple-600 text-white px-3 py-2 rounded text-sm hover:bg-purple-700 transition-colors">

📈 Regional Comparison

</button>

</div>

</div>

</div>

`;

html += `</div>`;

content.innerHTML = html;

// Show the panel

panel.classList.remove('translate-x-full');

panel.classList.add('translate-x-0');

console.log('LAD info panel displayed successfully');

}

// Helper function to find manure data for a LAD

function findManureDataForLAD(ladName, ladCode) {

if (!window.ladManureData) {

console.log('No manure data available in window.ladManureData');

return null;

}

console.log(`Searching for LAD: ${ladName} (${ladCode}) in ${window.ladManureData.length} records`);

// Use exact field definitions - no guessing

const match = window.ladManureData.find(entry => entry.lad\_code === ladCode);

if (match) {

console.log('Found exact code match:', match);

return match;

}

console.log('No manure data match found for:', ladName, ladCode);

console.log('Sample of available data:', window.ladManureData.slice(0, 3));

return null;

}

// Helper function to find manure data for an LPA

function findManureDataForLPA(lpaName, lpaCode) {

if (!window.lpaManureData) return null;

// Use exact field definitions - no guessing

return window.lpaManureData.find(entry => entry.lpa\_code === lpaCode);

}

// Generate manure breakdown for LAD

function generateManureBreakdown(manureEntry) {

const categories = [

{ name: '🐄 Beef FYM', tonnes: 'Beef FYM\_tonnes', gwh: 'Beef FYM\_gwh', color: '#8B4513' },

{ name: '🐄 Beef Slurry', tonnes: 'Beef Slurry\_tonnes', gwh: 'Beef Slurry\_gwh', color: '#A0522D' },

{ name: '🥛 Dairy FYM', tonnes: 'Dairy FYM\_tonnes', gwh: 'Dairy FYM\_gwh', color: '#DEB887' },

{ name: '🥛 Dairy Slurry', tonnes: 'Dairy Slurry\_tonnes', gwh: 'Dairy Slurry\_gwh', color: '#F4A460' },

{ name: '🐔 Broilers', tonnes: 'Broilers\_tonnes', gwh: 'Broilers\_gwh', color: '#FFD700' },

{ name: '🥚 Layers', tonnes: 'Layers\_tonnes', gwh: 'Layers\_gwh', color: '#FFA500' },

{ name: '🐷 Pigs', tonnes: 'Pigs\_tonnes', gwh: 'Pigs\_gwh', color: '#FFB6C1' },

{ name: '🐑 Sheep', tonnes: 'Sheep\_tonnes', gwh: 'Sheep\_gwh', color: '#F5F5DC' }

];

return categories.map(cat => {

// Handle both string and number values for tonnes and gwh

const tonnesValue = manureEntry[cat.tonnes];

const gwhValue = manureEntry[cat.gwh];

const tonnes = parseFloat(

typeof tonnesValue === 'string' ? tonnesValue.replace(/,/g, '') : tonnesValue || 0

);

const gwh = parseFloat(

typeof gwhValue === 'string' ? gwhValue.replace(/,/g, '') : gwhValue || 0

);

if (tonnes === 0) return '';

return `

<div class="flex justify-between items-center py-2 px-3 bg-gray-50 rounded">

<div class="flex items-center">

<div class="w-3 h-3 rounded-full mr-2" style="background-color: ${cat.color}"></div>

<span class="text-sm">${cat.name}</span>

</div>

<div class="text-right">

<div class="font-medium">${formatNumber(tonnes)} t/year</div>

<div class="text-xs text-gray-600">${formatNumber(gwh)} GWh</div>

</div>

</div>

`;

}).filter(html => html).join('');

}

// Calculate total tonnes for LPA (sum of all livestock categories)

function calculateLPATotalTonnes(lpaEntry) {

const categories = [

'B\_FYM\_Gras\_t', 'B\_FYM\_AraW\_t', 'B\_FYM\_AraS\_t', 'B\_Slu\_Gras\_t', 'B\_Slu\_AraS\_t', 'B\_Slu\_AraW\_t', 'B\_Grazing\_t',

'Bro\_FYM\_Gr\_t', 'Bro\_FYM\_Ar\_t', 'Bro\_FYM\_\_1\_t',

'D\_FYM\_Gras\_t', 'D\_FYM\_AraW\_t', 'D\_FYM\_AraS\_t', 'D\_Slu\_Gras\_t', 'D\_Slu\_AraS\_t', 'D\_Slu\_AraW\_t', 'D\_Grazing\_t',

'L\_FYM\_Gras\_t', 'L\_FYM\_AraW\_t', 'L\_FYM\_AraS\_t', 'L\_FrRngGra\_t',

'P\_OutGrazi\_t', 'P\_FYM\_Gras\_t', 'P\_FYM\_AraW\_t', 'P\_FYM\_AraS\_t', 'P\_Slu\_Gras\_t', 'P\_Slu\_AraS\_t', 'P\_Slu\_AraW\_t',

'S\_FYM\_Gras\_t', 'S\_FYM\_AraW\_t', 'S\_FYM\_AraS\_t', 'S\_Grazing\_t'

];

return categories.reduce((sum, field) => {

return sum + (parseFloat(lpaEntry[field]) || 0);

}, 0);

}

// Calculate estimated GWh for LPA (simplified estimation)

function calculateLPATotalGWh(lpaEntry) {

const totalTonnes = calculateLPATotalTonnes(lpaEntry);

// Simple estimation: ~1.1 GWh per 1000 tonnes (based on typical conversion rates)

return totalTonnes \* 0.0011;

}

// Generate manure breakdown for LPA (grouped by livestock type)

function generateLPAManureBreakdown(lpaEntry) {

const categories = [

{

name: '🐄 Beef Cattle',

color: '#8B4513',

fields: ['B\_FYM\_Gras\_t', 'B\_FYM\_AraW\_t', 'B\_FYM\_AraS\_t', 'B\_Slu\_Gras\_t', 'B\_Slu\_AraS\_t', 'B\_Slu\_AraW\_t', 'B\_Grazing\_t']

},

{

name: '🐔 Broilers',

color: '#FFD700',

fields: ['Bro\_FYM\_Gr\_t', 'Bro\_FYM\_Ar\_t', 'Bro\_FYM\_\_1\_t']

},

{

name: '🥛 Dairy Cattle',

color: '#DEB887',

fields: ['D\_FYM\_Gras\_t', 'D\_FYM\_AraW\_t', 'D\_FYM\_AraS\_t', 'D\_Slu\_Gras\_t', 'D\_Slu\_AraS\_t', 'D\_Slu\_AraW\_t', 'D\_Grazing\_t']

},

{

name: '🥚 Layers',

color: '#FFA500',

fields: ['L\_FYM\_Gras\_t', 'L\_FYM\_AraW\_t', 'L\_FYM\_AraS\_t', 'L\_FrRngGra\_t']

},

{

name: '🐷 Pigs',

color: '#FFB6C1',

fields: ['P\_OutGrazi\_t', 'P\_FYM\_Gras\_t', 'P\_FYM\_AraW\_t', 'P\_FYM\_AraS\_t', 'P\_Slu\_Gras\_t', 'P\_Slu\_AraS\_t', 'P\_Slu\_AraW\_t']

},

{

name: '🐑 Sheep',

color: '#F5F5DC',

fields: ['S\_FYM\_Gras\_t', 'S\_FYM\_AraW\_t', 'S\_FYM\_AraS\_t', 'S\_Grazing\_t']

}

];

return categories.map(cat => {

const total = cat.fields.reduce((sum, field) => {

return sum + (parseFloat(lpaEntry[field]) || 0);

}, 0);

if (total === 0) return '';

const estimatedGWh = total \* 0.0011; // Simple estimation

return `

<div class="flex justify-between items-center py-2 px-3 bg-gray-50 rounded">

<div class="flex items-center">

<div class="w-3 h-3 rounded-full mr-2" style="background-color: ${cat.color}"></div>

<span class="text-sm">${cat.name}</span>

</div>

<div class="text-right">

<div class="font-medium">${formatNumber(Math.round(total))} t/year</div>

<div class="text-xs text-gray-600">${formatNumber(Math.round(estimatedGWh \* 10) / 10)} GWh</div>

</div>

</div>

`;

}).filter(html => html).join('');

}

function showLADErrorPanel(ladName, errorMessage) {

const panel = document.getElementById('info-panel');

const title = document.getElementById('info-panel-title');

const content = document.getElementById('info-panel-content');

if (!panel) return;

if (title) title.textContent = `⚠️ Error Loading ${ladName}`;

if (content) {

content.innerHTML = `

<div class="error-panel bg-red-50 border border-red-200 rounded-lg p-4">

<p class="text-red-700 mb-2">Failed to load data for this Local Authority District.</p>

<p class="text-red-600 text-sm mb-4">${errorMessage}</p>

<button onclick="location.reload()" class="bg-red-600 text-white px-4 py-2 rounded hover:bg-red-700 transition-colors">

Refresh Page

</button>

</div>

`;

}

panel.classList.remove('translate-x-full');

panel.classList.add('translate-x-0');

}

function togglePlantsContent(button) {

try {

const contentElement = button.closest('.bg-white')?.querySelector('.plants-content');

if (contentElement && contentElement.style) {

contentElement.style.display = contentElement.style.display === 'none' ? 'block' : 'none';

}

} catch (error) {

console.warn('Error toggling plants content:', error);

}

}

function toggleStatusGroup(statusId) {

const element = document.getElementById(`group-${statusId}`);

if (element && element.style) {

element.style.display = element.style.display === 'none' ? 'block' : 'none';

}

}

function toggleManureSection() {

const element = document.getElementById('manure-content');

if (element && element.style) {

element.style.display = element.style.display === 'none' ? 'block' : 'none';

}

}

function toggleManureContent(button) {

try {

const contentElement = button.closest('.bg-white')?.querySelector('.manure-content');

if (contentElement && contentElement.style) {

contentElement.style.display = contentElement.style.display === 'none' ? 'block' : 'none';

}

} catch (error) {

console.warn('Error toggling manure content:', error);

}

}

function closeInfoPanel() {

const panel = document.getElementById('info-panel');

if (panel) {

panel.classList.add('translate-x-full');

panel.classList.remove('translate-x-0');

panel.classList.remove('open');

}

// Reset LAD visual styling

try {

map.setPaintProperty('lad-fill', 'fill-opacity', 0.06);

map.setPaintProperty('lad', 'line-width', 1.5);

} catch (error) {

console.warn('Failed to reset LAD styling:', error);

}

}

// ==================== DATA LOADING FUNCTIONS ====================

async function loadCSVData(url, description) {

try {

const response = await fetch(url);

if (!response.ok) {

throw new Error(`HTTP ${response.status}: ${response.statusText}`);

}

const csvText = await response.text();

// Debug: Check CSV header line for field structure

if (description === 'AD Plants data') {

const lines = csvText.split('\n');

const headerLine = lines[0];

console.log('🔍 CSV HEADER LINE:', headerLine);

console.log('🔍 CSV HEADER FIELDS:', headerLine.split(',').length);

// Check if DNO and WaterCo are in the header

const headers = headerLine.split(',');

console.log('🔍 HEADER INCLUDES DNO:', headers.includes('DNO'));

console.log('🔍 HEADER INCLUDES WaterCo:', headers.includes('WaterCo'));

console.log('🔍 LAST 10 HEADERS:', headers.slice(-10));

}

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

Papa.parse(csvText, {

header: true,

dynamicTyping: false, // Keep all fields as strings to avoid parsing issues

skipEmptyLines: true,

trimHeaders: true, // Clean header field names

transform: (value, field) => {

// Clean up field values, especially for DNO and WaterCo

if (typeof value === 'string') {

return value.trim();

}

return value;

},

complete: (results) => {

if (results.errors && results.errors.length > 0) {

console.warn(`Parsing warnings for ${description}:`, results.errors);

}

// Debug: Log sample data structure for AD plants

if (description === 'AD Plants data' && results.data.length > 0) {

console.log('🔍 AD PLANTS CSV DEBUGGING:');

console.log('Total plants loaded:', results.data.length);

// Check first few plants for field structure

const sample = results.data[0];

console.log('First plant fields:', Object.keys(sample));

console.log('Sample first plant:', sample);

// Look for Wyke Farms specifically

const wyke = results.data.find(row => row['Site name'] && row['Site name'].includes('Wyke Farms'));

if (wyke) {

console.log('🎯 WYKE FARMS PLANT FOUND:');

console.log('Wyke Farms full data:', wyke);

console.log('DNO field value:', wyke.DNO);

console.log('WaterCo field value:', wyke.WaterCo);

}

// Look for Garth Road specifically from the user's test

const garth = results.data.find(row => row['Site name'] && row['Site name'].includes('Garth Road'));

if (garth) {

console.log('🎯 GARTH ROAD PLANT FOUND:');

console.log('Garth Road full data:', garth);

console.log('DNO field value:', garth.DNO);

console.log('WaterCo field value:', garth.WaterCo);

}

// Look for Hibaldstow Airfield specifically from console logs

const hibald = results.data.find(row => row['Site name'] && row['Site name'].includes('Hibaldstow'));

if (hibald) {

console.log('🎯 HIBALDSTOW AIRFIELD PLANT FOUND:');

console.log('Hibaldstow full data:', hibald);

console.log('DNO field value:', hibald.DNO);

console.log('WaterCo field value:', hibald.WaterCo);

}

}

resolve(results.data);

},

error: (error) => {

console.error(`Failed to parse ${description}:`, error);

reject(new Error(`Failed to parse ${description}: ${error.message}`));

}

});

});

} catch (error) {

console.error(`Failed to load ${description}:`, error);

throw new Error(`Failed to load ${description}: ${error.message}`);

}

}

async function loadTopoJSONData(url, description) {

try {

const response = await fetch(url);

if (!response.ok) {

throw new Error(`HTTP ${response.status}: ${response.statusText}`);

}

const topology = await response.json();

console.log(`TopoJSON loaded for ${description}, objects:`, Object.keys(topology.objects || {}));

if (!topology.objects || Object.keys(topology.objects).length === 0) {

throw new Error('No objects found in TopoJSON');

}

// Use the first available object key

const objectKey = Object.keys(topology.objects)[0];

const feature = topojson.feature(topology, topology.objects[objectKey]);

console.log(`Successfully processed ${description} with object: ${objectKey}, features: ${feature.features?.length || 0}`);

return { topology, feature, objectKey };

} catch (error) {

console.error(`Failed to load TopoJSON from ${url}:`, error);

throw new Error(`Failed to load ${description}: ${error.message}`);

}

}

async function loadKMLData(url) {

try {

const response = await fetch(url);

if (!response.ok) {

throw new Error(`HTTP ${response.status}: ${response.statusText}`);

}

const kmlText = await response.text();

const parser = new DOMParser();

const kmlDoc = parser.parseFromString(kmlText, 'text/xml');

const placemarks = kmlDoc.getElementsByTagName('Placemark');

const features = [];

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

const placemark = placemarks[i];

const coordinates = placemark.getElementsByTagName('coordinates')[0]?.textContent?.trim();

if (coordinates) {

const coords = coordinates.split(' ').map(coord => {

const [lng, lat] = coord.split(',').map(Number);

return [lng, lat];

});

features.push({

type: 'Feature',

properties: {},

geometry: {

type: 'LineString',

coordinates: coords

}

});

}

}

return {

type: 'FeatureCollection',

features

};

} catch (error) {

console.error(`Failed to load KML from ${url}:`, error);

throw new Error(`Failed to load KML data: ${error.message}`);

}

}

// ==================== LAYER MANAGER CLASS ====================

class LayerManager {

constructor(map) {

this.map = map;

this.registry = new Map();

this.layers = {}; // Add missing layers property for compatibility

this.loadedSources = new Set();

this.activeLayers = new Set(['lad']);

}

async addLayer(layerConfig) {

const { id, source, type, paint, layout, filter, beforeId } = layerConfig;

try {

// Add source if not already loaded and doesn't exist in map

if (!this.loadedSources.has(source.id) && !this.map.getSource(source.id)) {

this.map.addSource(source.id, source.config);

this.loadedSources.add(source.id);

}

// Skip if layer already exists

if (this.map.getLayer(id)) {

console.warn(`Layer ${id} already exists, skipping`);

return true;

}

// Add layer

const layerSpec = {

id,

type,

source: source.id,

paint: paint || {},

layout: layout || { visibility: 'none' }

};

if (filter) layerSpec.filter = filter;

if (source.sourceLayer) layerSpec['source-layer'] = source.sourceLayer;

// Add layer with beforeId parameter to control z-index

// For AD plant layers, ensure they're added at the very top

if (beforeId) {

this.map.addLayer(layerSpec, beforeId);

} else if (id.startsWith('ad-')) {

// AD plants always go to the absolute top - no beforeId ensures this

this.map.addLayer(layerSpec);

} else {

this.map.addLayer(layerSpec);

}

this.registry.set(id, layerConfig);

this.layers[id] = layerConfig; // Add to layers object for compatibility

return true;

} catch (error) {

console.error(`Failed to add layer ${id}:`, error);

return false;

}

}

toggle(layerId, visible) {

console.log(`Toggling AD layer: ${layerId}`);

console.log('Available layers in registry:', Array.from(this.registry.keys()));

if (!this.registry.has(layerId) && !this.layers[layerId]) {

console.warn(`Layer ${layerId} not found in registry`);

return;

}

try {

this.map.setLayoutProperty(layerId, 'visibility', visible ? 'visible' : 'none');

if (visible) {

this.activeLayers.add(layerId);

console.log('Active layers:', Array.from(this.activeLayers));

// Debug: Check if layer is actually visible on the map for brownfield

if (layerId === 'brownfield') {

setTimeout(() => {

console.log(`Brownfield layer visibility after toggle: ${this.map.getLayoutProperty(layerId, 'visibility')}`);

console.log(`Brownfield layer paint:`, {

'fill-color': this.map.getPaintProperty(layerId, 'fill-color'),

'fill-opacity': this.map.getPaintProperty(layerId, 'fill-opacity'),

'fill-outline-color': this.map.getPaintProperty(layerId, 'fill-outline-color')

});

console.log(`Map has brownfield source:`, !!this.map.getSource(layerId));

const source = this.map.getSource(layerId);

if (source && source.\_data) {

console.log(`Brownfield source has ${source.\_data.features ? source.\_data.features.length : 0} features`);

const sampleFeature = source.\_data.features ? source.\_data.features[0] : null;

if (sampleFeature) {

console.log(`Sample brownfield feature geometry type:`, sampleFeature.geometry.type);

console.log(`Sample brownfield properties:`, Object.keys(sampleFeature.properties));

}

}

// Force map repaint to ensure layer renders

this.map.triggerRepaint();

console.log('Forced map repaint for brownfield layer');

}, 100);

}

} else {

this.activeLayers.delete(layerId);

console.log('Active layers:', Array.from(this.activeLayers));

}

updateLegend();

} catch (error) {

console.error(`Failed to toggle layer ${layerId}:`, error);

}

}

isVisible(layerId) {

try {

return this.map.getLayoutProperty(layerId, 'visibility') === 'visible';

} catch {

return false;

}

}

getActiveLayers() {

return new Set(this.activeLayers);

}

hasLayer(layerId) {

return this.registry.has(layerId);

}

updateSourceData(sourceId, data) {

try {

if (this.map.getSource(sourceId)) {

this.map.getSource(sourceId).setData(data);

}

} catch (error) {

console.error(`Failed to update source ${sourceId}:`, error);

}

}

uncheckAllLayers() {

for (const layerId of this.activeLayers) {

if (layerId !== 'lad') { // Keep LAD always visible

this.toggle(layerId, false);

}

}

}

}

// ==================== ON-DEMAND LOADING FUNCTIONS ====================

async function loadADPlantsLayers() {

if (adLayersLoaded) return;

try {

console.log('Loading AD Plants data and layers...');

// Load AD plants data first

adPlantsData = await loadCSVData(URLS.AD\_CSV, 'AD Plants data');

console.log('AD Plants data loaded:', adPlantsData.length);

// Initialize layers

await initializeADPlantsLayers();

// Update UI

updatePlantsOverview();

initializeSearch();

updateADPlantsControls();

adLayersLoaded = true;

console.log('AD Plants layers loaded successfully');

} catch (error) {

console.error('Failed to load AD Plants layers:', error);

showToast('Failed to load AD Plants data', 'error');

}

}

async function loadManureLayers() {

if (manureLayersLoaded) return;

try {

console.log('Loading Manure volume layers...');

// Load manure volumes TopJSON data directly

const manureResult = await loadTopoJSONData(MANURE\_VOLUMES\_URL, 'Manure volumes TopJSON');

manureVolumesData = manureResult.feature;

console.log('Manure volumes TopJSON data loaded:', manureVolumesData?.features?.length, 'features');

// Debug the field structure and auto-update mappings

if (manureVolumesData?.features?.length > 0) {

console.log('Sample feature for manual inspection:', manureVolumesData.features[0]);

// Manual field mapping update

const sampleFeature = manureVolumesData.features[0];

const allFields = Object.keys(sampleFeature.properties || {});

console.log('Manually updating field mappings with discovered fields:', allFields.slice(0, 10));

updateManureFieldMappings(allFields);

}

// Initialize manure source first

if (!map.getSource('manure-volumes-source')) {

map.addSource('manure-volumes-source', {

type: 'geojson',

data: {

type: 'FeatureCollection',

features: []

}

});

}

// Initialize manure volume layers using TopJSON data

try {

await renderManureVolumesLayer();

} catch (renderError) {

console.error('Failed to render manure layers:', renderError);

console.log('Skipping manure layer rendering and continuing with UI setup');

}

// Update UI with new manure categories

console.log('Calling correct updateManureControls function');

updateManureControls();

manureLayersLoaded = true;

console.log('Manure layers loaded successfully');

} catch (error) {

console.error('Failed to load Manure layers:', error);

showToast('Failed to load Manure volumes data', 'error');

}

}

async function renderManureVolumesLayer() {

if (!manureVolumesData) return;

try {

console.log('Rendering manure volume layers...');

// Get active categories

const activeCategories = Object.keys(manureVolumesVisibility).filter(

category => manureVolumesVisibility[category]

);

if (activeCategories.length === 0) {

// Hide all manure layers when none are selected

if (map.getLayer('manure-volumes-dynamic')) {

map.setLayoutProperty('manure-volumes-dynamic', 'visibility', 'none');

}

return;

}

// First pass: collect data values separately for each category

const categoryDataValues = {};

activeCategories.forEach(category => {

categoryDataValues[category] = [];

});

manureVolumesData.features.forEach(feature => {

activeCategories.forEach(category => {

const result = aggregateManureVolumes(feature.properties, category);

if (result.total > 0) {

categoryDataValues[category].push(result.total);

}

});

});

// Log data range for each category (converting kg to tonnes for readability)

activeCategories.forEach(category => {

if (categoryDataValues[category].length > 0) {

const min = Math.min(...categoryDataValues[category]);

const max = Math.max(...categoryDataValues[category]);

const minTonnes = Math.round(min / 1000);

const maxTonnes = Math.round(max / 1000);

// Calculate statistical distribution for debugging

const mean = categoryDataValues[category].reduce((sum, val) => sum + val, 0) / categoryDataValues[category].length;

const variance = categoryDataValues[category].reduce((sum, val) => sum + Math.pow(val - mean, 2), 0) / categoryDataValues[category].length;

const stdDev = Math.sqrt(variance);

console.log(`${category} data range: ${minTonnes.toLocaleString()} - ${maxTonnes.toLocaleString()} tonnes (${min.toLocaleString()} - ${max.toLocaleString()} kg raw) across ${categoryDataValues[category].length} hexagons`);

console.log(`${category} statistical distribution:`);

console.log(` Mean: ${Math.round(mean / 1000).toLocaleString()} tonnes`);

console.log(` Std Dev: ${Math.round(stdDev / 1000).toLocaleString()} tonnes`);

// Calculate statistical percentile boundaries (in tonnes)

const zScores = [-2, -0.84, -0.25, 0.25, 0.52, 0.84, 1.28, 2];

const boundaries = zScores.map(z => Math.max(0, Math.round((mean + z \* stdDev) / 1000)));

console.log(`${category} statistical tiers (z-score based):`);

console.log(` Tier 1-2 (bottom 20%, z < -0.84): 0 - ${boundaries[1]} tonnes`);

console.log(` Tier 3 (20-40%, z -0.84 to -0.25): ${boundaries[1]} - ${boundaries[2]} tonnes`);

console.log(` Tier 4 (40-60%, z -0.25 to 0.25): ${boundaries[2]} - ${boundaries[3]} tonnes`);

console.log(` Tier 5 (60-70%, z 0.25 to 0.52): ${boundaries[3]} - ${boundaries[4]} tonnes`);

console.log(` Tier 6 (70-80%, z 0.52 to 0.84): ${boundaries[4]} - ${boundaries[5]} tonnes`);

console.log(` Tier 7 (80-90%, z 0.84 to 1.28): ${boundaries[5]} - ${boundaries[6]} tonnes`);

console.log(` Tier 8 (top 10%, z > 1.28): ${boundaries[6]}+ tonnes`);

}

});

// Process features for visualization

const visualizationFeatures = manureVolumesData.features.map(feature => {

// Find the category with highest volume for this polygon

let maxVolume = 0;

let dominantCategory = null;

activeCategories.forEach(category => {

const result = aggregateManureVolumes(feature.properties, category);

if (result.total > maxVolume) {

maxVolume = result.total;

dominantCategory = category;

}

});

if (!dominantCategory || maxVolume === 0) {

return {

...feature,

properties: {

...feature.properties,

fillColor: 'rgba(0,0,0,0)',

fillOpacity: 0

}

};

}

const colorData = getManureIntensityColor(maxVolume, dominantCategory, categoryDataValues[dominantCategory]);

const fillColor = `rgba(${colorData.r}, ${colorData.g}, ${colorData.b}, ${colorData.opacity})`;

return {

...feature,

properties: {

...feature.properties,

fillColor: fillColor,

fillOpacity: colorData.opacity,

dominantCategory: dominantCategory,

totalVolume: maxVolume

}

};

});

// Add manure source if it doesn't exist

if (!map.getSource('manure-volumes-source')) {

map.addSource('manure-volumes-source', {

type: 'geojson',

data: {

type: 'FeatureCollection',

features: visualizationFeatures

}

});

} else {

// Update existing source

map.getSource('manure-volumes-source').setData({

type: 'FeatureCollection',

features: visualizationFeatures

});

}

// Add dynamic manure layer for visualization

if (!map.getLayer('manure-volumes-dynamic')) {

map.addLayer({

id: 'manure-volumes-dynamic',

source: 'manure-volumes-source',

type: 'fill',

paint: {

'fill-color': ['get', 'fillColor'],

'fill-opacity': 1.0

},

layout: {

visibility: 'visible'

}

}); // Add to top of layer stack

// Add click handler for manure polygons

map.on('click', 'manure-volumes-dynamic', (e) => {

if (e.features && e.features.length > 0) {

e.preventDefault();

e.originalEvent.stopPropagation();

const feature = e.features[0];

console.log('Manure hexagon clicked:', feature.properties);

showManureInfoPanel(feature.properties, e.lngLat);

}

});

map.on('mouseenter', 'manure-volumes-dynamic', () => {

map.getCanvas().style.cursor = 'pointer';

});

map.on('mouseleave', 'manure-volumes-dynamic', () => {

map.getCanvas().style.cursor = '';

});

} else {

// Show the layer

map.setLayoutProperty('manure-volumes-dynamic', 'visibility', 'visible');

}

// Register layer with LayerManager

layerManager.registry.set('manure-volumes-dynamic', true);

layerManager.layers['manure-volumes-dynamic'] = true;

console.log('Manure volume layers rendered successfully');

} catch (error) {

console.error('Error rendering manure volume layers:', error);

throw error;

}

}

function updateManureDisplay() {

try {

console.log('updateManureDisplay called. Data check:', {

hasData: !!manureVolumesData,

hasFeatures: !!(manureVolumesData?.features),

featureCount: manureVolumesData?.features?.length

});

if (!manureVolumesData || !manureVolumesData.features) {

console.log('Manure data not ready for display update');

return;

}

console.log('Updating manure display - data ready with', manureVolumesData.features.length, 'features');

// Update visibility flags

Object.keys(manureVolumesVisibility).forEach(category => {

const categoryId = category.toLowerCase().replace(/\s+/g, '-');

const checkbox = document.getElementById(`manure-${categoryId}`);

if (checkbox) {

manureVolumesVisibility[category] = checkbox.checked;

}

});

console.log('Updated manure visibility:', manureVolumesVisibility);

// Re-render the manure layer with new visibility settings

renderManureVolumesLayer();

} catch (error) {

console.error('Error updating manure display:', error);

}

}

function showManureInfoPanel(properties, lngLat) {

const infoPanel = document.getElementById('info-panel');

const infoPanelTitle = document.getElementById('info-panel-title');

const infoPanelContent = document.getElementById('info-panel-content');

console.log('showManureInfoPanel called with:', {

panel: !!infoPanel,

title: !!infoPanelTitle,

content: !!infoPanelContent,

properties: properties

});

if (!infoPanel || !infoPanelTitle || !infoPanelContent) {

console.error('Info panel elements not found:', {

panel: !!infoPanel,

title: !!infoPanelTitle,

content: !!infoPanelContent

});

return;

}

// Convert kg to tonnes for display (divide by 1000)

const volumeInTonnes = properties.totalVolume ? Math.round(properties.totalVolume / 1000) : 0;

// Update title

infoPanelTitle.textContent = 'Manure Volume Data';

// Show basic polygon information

let html = `

<div class="space-y-4">

<div class="bg-slate-50 p-3 rounded">

<div class="grid grid-cols-1 gap-2 text-sm">

<div><strong>Grid ID:</strong> ${properties.GridID || properties.Grid\_ID || 'Unknown'}</div>

<div><strong>Dominant Category:</strong> ${properties.dominantCategory || 'Unknown'}</div>

<div><strong>Volume:</strong> ${volumeInTonnes ? volumeInTonnes.toLocaleString() + ' tonnes' : 'N/A'}</div>

<div><strong>Raw Volume:</strong> ${properties.totalVolume ? properties.totalVolume.toLocaleString() + ' kg' : 'N/A'}</div>

</div>

</div>

</div>

`;

infoPanelContent.innerHTML = html;

// Show the panel

infoPanel.classList.add('open');

infoPanel.style.transform = 'translateX(0)';

}

// This function was moved to line 1705 - using the correct container selector

// Individual environmental layer loading functions for on-demand loading

async function loadAONBLayer() {

if (layerManager.hasLayer('aonb')) return;

try {

console.log('📥 Loading Areas of Outstanding Natural Beauty on demand...');

const { feature: geoJson } = await loadTopoJSONData(URLS.AONB\_TOPO, 'Areas of Outstanding Natural Beauty');

console.log(`AONB loaded: ${geoJson.features.length} features`);

await layerManager.addLayer({

id: 'aonb',

source: {

id: 'aonb',

config: {

type: 'geojson',

data: geoJson

}

},

type: 'fill',

paint: {

'fill-color': '#22c55e',

'fill-opacity': 0.3,

'fill-outline-color': '#22c55e'

},

layout: { visibility: 'none' }

});

setupLayerEventHandlers('aonb', 'Areas of Outstanding Natural Beauty');

console.log('✅ AONB layer loaded on demand');

} catch (error) {

console.error('❌ Failed to load AONB layer:', error);

showToast('Failed to load Areas of Outstanding Natural Beauty', 'error');

}

}

async function loadSSSILayer() {

if (layerManager.hasLayer('sssi')) return;

try {

console.log('📥 Loading Sites of Special Scientific Interest on demand...');

const { feature: geoJson } = await loadTopoJSONData(URLS.SSSI\_TOPO, 'Sites of Special Scientific Interest');

console.log(`SSSI loaded: ${geoJson.features.length} features`);

await layerManager.addLayer({

id: 'sssi',

source: {

id: 'sssi',

config: {

type: 'geojson',

data: geoJson

}

},

type: 'fill',

paint: {

'fill-color': '#a855f7',

'fill-opacity': 0.3,

'fill-outline-color': '#a855f7'

},

layout: { visibility: 'none' }

});

setupLayerEventHandlers('sssi', 'Sites of Special Scientific Interest');

console.log('✅ SSSI layer loaded on demand');

} catch (error) {

console.error('❌ Failed to load SSSI layer:', error);

showToast('Failed to load Sites of Special Scientific Interest', 'error');

}

}

async function loadNVZLayer() {

if (layerManager.hasLayer('nvz')) return;

try {

console.log('📥 Loading Nitrate Vulnerable Zones on demand...');

const { feature: geoJson } = await loadTopoJSONData(URLS.NVZ\_TOPO, 'Nitrate Vulnerable Zones');

console.log(`NVZ loaded: ${geoJson.features.length} features`);

await layerManager.addLayer({

id: 'nvz',

source: {

id: 'nvz',

config: {

type: 'geojson',

data: geoJson

}

},

type: 'fill',

paint: {

'fill-color': '#f59e0b',

'fill-opacity': 0.3,

'fill-outline-color': '#f59e0b'

},

layout: { visibility: 'none' }

});

setupLayerEventHandlers('nvz', 'Nitrate Vulnerable Zones');

console.log('✅ NVZ layer loaded on demand');

} catch (error) {

console.error('❌ Failed to load NVZ layer:', error);

showToast('Failed to load Nitrate Vulnerable Zones', 'error');

}

}

async function loadFloodLayer() {

if (layerManager.hasLayer('flood')) return;

try {

console.log('📥 Loading Historic Flood Zones on demand...');

const { feature: geoJson } = await loadTopoJSONData(URLS.FLOOD\_TOPO, 'Historic Flood Zones');

console.log(`Flood zones loaded: ${geoJson.features.length} features`);

await layerManager.addLayer({

id: 'flood',

source: {

id: 'flood',

config: {

type: 'geojson',

data: geoJson

}

},

type: 'fill',

paint: {

'fill-color': '#3b82f6',

'fill-opacity': 0.3,

'fill-outline-color': '#3b82f6'

},

layout: { visibility: 'none' }

});

setupLayerEventHandlers('flood', 'Historic Flood Zones');

console.log('✅ Flood layer loaded on demand');

} catch (error) {

console.error('❌ Failed to load Flood layer:', error);

showToast('Failed to load Historic Flood Zones', 'error');

}

}

async function loadEnvironmentalLayers() {

if (environmentalLayersLoaded) return;

console.log('🔄 Initializing Environmental layers section (on-demand loading)...');

// Mark as initialized but don't load any layers yet

environmentalLayersLoaded = true;

console.log('✅ Environmental layers section ready - layers will load on-demand when toggled');

}

// Helper function to determine layer configuration

function determineLayerConfiguration(geoJson, layerConfig) {

let layerType = 'fill';

let paintConfig = {

'fill-color': layerConfig.color,

'fill-opacity': 0.3,

'fill-outline-color': layerConfig.color

};

if (geoJson.features && geoJson.features.length > 0) {

const geometryType = geoJson.features[0].geometry?.type;

console.log(`${layerConfig.name} - Geometry type: ${geometryType}`);

if (geometryType === 'Point' || geometryType === 'MultiPoint') {

layerType = 'circle';

paintConfig = {

'circle-radius': 4,

'circle-color': layerConfig.color,

'circle-opacity': 0.8,

'circle-stroke-width': 1,

'circle-stroke-color': '#ffffff'

};

console.log(`${layerConfig.name} - Using circle layer for point geometry`);

}

}

return { layerType, paintConfig };

}

// Helper function to sanitize properties for safe display

function sanitizeProperties(properties) {

if (!properties || typeof properties !== 'object') {

return {};

}

const sanitized = {};

Object.keys(properties).forEach(key => {

const value = properties[key];

if (typeof value === 'string') {

// Basic HTML sanitization - remove potentially dangerous tags

sanitized[key] = value

.replace(/<script\b[^<]\*(?:(?!<\/script>)<[^<]\*)\*<\/script>/gi, '')

.replace(/<[^>]+>/g, '')

.slice(0, 1000); // Limit length

} else if (typeof value === 'number' || typeof value === 'boolean') {

sanitized[key] = value;

} else {

sanitized[key] = String(value).slice(0, 500);

}

});

return sanitized;

}

// Add global error handlers for unhandled promise rejections

window.addEventListener('unhandledrejection', (event) => {

console.error('🚨 Unhandled promise rejection detected:', event);

// Extract more information about the error

let errorInfo = {

timestamp: new Date().toISOString(),

eventType: 'unhandledrejection'

};

if (event.reason) {

if (event.reason instanceof Error) {

errorInfo.message = event.reason.message;

errorInfo.stack = event.reason.stack;

errorInfo.name = event.reason.name;

} else if (typeof event.reason === 'string') {

errorInfo.message = event.reason;

} else if (typeof event.reason === 'object') {

errorInfo.reasonObject = JSON.stringify(event.reason, null, 2);

} else {

errorInfo.reasonType = typeof event.reason;

errorInfo.reasonValue = String(event.reason);

}

} else {

errorInfo.message = 'Promise rejected with no reason';

}

console.error('Detailed promise rejection info:', errorInfo);

// Try to show user-friendly error only for significant errors

if (errorInfo.message && !errorInfo.message.includes('AbortError')) {

try {

showToast('Data loading interrupted - please try again', 'error');

} catch (toastError) {

console.warn('Could not show error toast:', toastError);

}

}

// Prevent default browser console error

event.preventDefault();

});

// Add global error handler for other script errors

window.addEventListener('error', (event) => {

console.error('🚨 Global script error:', {

message: event.message,

filename: event.filename,

lineno: event.lineno,

colno: event.colno,

error: event.error,

timestamp: new Date().toISOString()

});

});

// Add cleanup on page unload

window.addEventListener('beforeunload', () => {

console.log('🧹 Cleaning up all event handlers before page unload...');

cleanupAllEventHandlers();

});

async function loadInfrastructureLayers() {

if (infrastructureLayersLoaded) return;

try {

console.log('Loading Infrastructure layers - only NTS initially...');

// Only load NTS initially when Infrastructure section opens

try {

const kmlData = await loadKMLData(URLS.NTS\_KML);

// Store NTS data globally for proximity analysis

window.ntsData = kmlData;

console.log('NTS data stored for proximity analysis:', kmlData.features?.length || 0, 'features');

await layerManager.addLayer({

id: 'nts',

source: {

id: 'nts',

config: {

type: 'geojson',

data: kmlData

}

},

type: 'line',

paint: {

'line-color': '#dc2626',

'line-width': 3,

'line-opacity': 0.8

},

layout: { visibility: 'none' }

});

console.log('NTS layer loaded successfully');

} catch (error) {

console.warn('Failed to load NTS layer:', error);

}

// NTS is loaded but not visible - user must toggle checkbox

infrastructureLayersLoaded = true;

console.log('Infrastructure section initialized - only NTS loaded');

} catch (error) {

console.error('Failed to load Infrastructure layers:', error);

showToast('Failed to load Infrastructure data', 'error');

}

}

// Individual infrastructure layer loading functions

async function loadDNOLayer() {

if (layerManager.hasLayer('dno')) return;

try {

console.log('Loading DNO layer on demand...');

const { feature: geoJson } = await loadTopoJSONData(URLS.DNO\_TOPO, 'Distribution Network Operators');

console.log(`DNO loaded: ${geoJson.features.length} features`);

await layerManager.addLayer({

id: 'dno',

source: {

id: 'dno',

config: {

type: 'geojson',

data: geoJson

}

},

type: 'line',

paint: {

'line-color': '#ef4444',

'line-width': 2,

'line-opacity': 0.8

},

layout: { visibility: 'visible' }

});

// Add click handlers

map.on('click', 'dno', (e) => {

if (e.features && e.features.length > 0) {

const feature = e.features[0];

showPolygonInfoPanel(feature.properties, 'Distribution Network Operators', '#ef4444', e.lngLat);

}

});

map.on('mouseenter', 'dno', () => {

map.getCanvas().style.cursor = 'pointer';

});

map.on('mouseleave', 'dno', () => {

map.getCanvas().style.cursor = '';

});

console.log('DNO layer loaded successfully');

} catch (error) {

console.warn('Failed to load DNO layer:', error);

}

}

async function loadWaterLayer() {

if (layerManager.hasLayer('water')) return;

try {

console.log('Loading Water layer on demand...');

const { feature: geoJson } = await loadTopoJSONData(URLS.WATER\_TOPO, 'Water Company Boundaries');

console.log(`Water loaded: ${geoJson.features.length} features`);

await layerManager.addLayer({

id: 'water',

source: {

id: 'water',

config: {

type: 'geojson',

data: geoJson

}

},

type: 'line',

paint: {

'line-color': '#06b6d4',

'line-width': 2,

'line-opacity': 0.8

},

layout: { visibility: 'visible' }

});

// Add click handlers

map.on('click', 'water', (e) => {

if (e.features && e.features.length > 0) {

const feature = e.features[0];

showPolygonInfoPanel(feature.properties, 'Water Company Boundaries', '#06b6d4', e.lngLat);

}

});

map.on('mouseenter', 'water', () => {

map.getCanvas().style.cursor = 'pointer';

});

map.on('mouseleave', 'water', () => {

map.getCanvas().style.cursor = '';

});

console.log('Water layer loaded successfully');

} catch (error) {

console.warn('Failed to load Water layer:', error);

}

}

async function loadBrownfieldLayer() {

if (layerManager.hasLayer('brownfield')) return;

try {

console.log('Loading Brownfield layer on demand...');

const { feature: geoJson } = await loadTopoJSONData(URLS.BROWNFIELD\_TOPO, 'Brownfield Land');

console.log(`Brownfield loaded: ${geoJson.features.length} features`);

await layerManager.addLayer({

id: 'brownfield',

source: {

id: 'brownfield',

config: {

type: 'geojson',

data: geoJson

}

},

type: 'circle',

paint: {

'circle-color': '#8B4513',

'circle-radius': 4,

'circle-opacity': 0.8,

'circle-stroke-color': '#654321',

'circle-stroke-width': 1

},

layout: { visibility: 'visible' }

});

// Add click handlers

map.on('click', 'brownfield', (e) => {

if (e.features && e.features.length > 0) {

const feature = e.features[0];

showPolygonInfoPanel(feature.properties, 'Brownfield Land', '#92400e', e.lngLat);

}

});

map.on('mouseenter', 'brownfield', () => {

map.getCanvas().style.cursor = 'pointer';

});

map.on('mouseleave', 'brownfield', () => {

map.getCanvas().style.cursor = '';

});

console.log('Brownfield layer loaded successfully');

} catch (error) {

console.warn('Failed to load Brownfield layer:', error);

}

}

async function loadAgriculturalLayers() {

if (agriculturalLayersLoaded) return;

try {

console.log('Loading Agricultural layers...');

const { feature: geoJson } = await loadTopoJSONData(URLS.ALC\_TOPO, 'Agricultural Land Classification');

// Log sample data to understand the structure and check both possible property names

console.log('Sample ALC features:', geoJson.features.slice(0, 5).map(f => ({

properties: f.properties,

grade\_caps: f.properties.ALC\_GRADE,

grade\_mixed: f.properties.ALC\_Grade,

all\_props: Object.keys(f.properties)

})));

// Get all unique grades from the data - try both property names

const uniqueGrades1 = [...new Set(geoJson.features.map(f => f.properties.ALC\_GRADE))].filter(Boolean);

const uniqueGrades2 = [...new Set(geoJson.features.map(f => f.properties.ALC\_Grade))].filter(Boolean);

console.log('Unique ALC\_GRADE in data:', uniqueGrades1);

console.log('Unique ALC\_Grade in data:', uniqueGrades2);

// Use whichever property exists

const gradeProperty = uniqueGrades1.length > 0 ? 'ALC\_GRADE' : 'ALC\_Grade';

console.log('Using property:', gradeProperty);

// Define ALC grade configurations based on actual data structure

const alcGrades = [

{ id: 'alc-grade-1', grade: 'Grade 1', color: '#22c55e', opacity: 0.6 }, // Green - Best quality

{ id: 'alc-grade-2', grade: 'Grade 2', color: '#84cc16', opacity: 0.6 }, // Lime green

{ id: 'alc-grade-3', grade: 'Grade 3', color: '#eab308', opacity: 0.5 }, // Yellow

{ id: 'alc-grade-4', grade: 'Grade 4', color: '#f97316', opacity: 0.4 }, // Orange

{ id: 'alc-grade-5', grade: 'Grade 5', color: '#ef4444', opacity: 0.4 } // Red - Poorest quality

];

// Create individual layers for each ALC grade

for (const gradeConfig of alcGrades) {

// Filter features by ALC grade - use detected property name

const filteredFeatures = {

...geoJson,

features: geoJson.features.filter(feature =>

feature.properties[gradeProperty] === gradeConfig.grade

)

};

await layerManager.addLayer({

id: gradeConfig.id,

source: {

id: gradeConfig.id,

config: {

type: 'geojson',

data: filteredFeatures

}

},

type: 'fill',

paint: {

'fill-color': gradeConfig.color,

'fill-opacity': gradeConfig.opacity,

'fill-outline-color': gradeConfig.color

},

layout: { visibility: 'none' }

});

// Add click handlers for ALC layers

map.on('click', gradeConfig.id, (e) => {

if (e.features && e.features.length > 0) {

const feature = e.features[0];

showPolygonInfoPanel(feature.properties, `Agricultural Land Classification - ${gradeConfig.grade}`, gradeConfig.color, e.lngLat);

}

});

map.on('mouseenter', gradeConfig.id, () => {

map.getCanvas().style.cursor = 'pointer';

});

map.on('mouseleave', gradeConfig.id, () => {

map.getCanvas().style.cursor = '';

});

console.log(`ALC ${gradeConfig.id} layer (${gradeConfig.grade}) loaded: ${filteredFeatures.features.length} features`);

// Debug: show a sample of filtered features

if (filteredFeatures.features.length > 0) {

console.log(`Sample features for ${gradeConfig.grade}:`, filteredFeatures.features.slice(0, 2).map(f => f.properties[gradeProperty]));

} else {

console.log(`No features found for "${gradeConfig.grade}". Checking all grades in data...`);

const allGradesInData = geoJson.features.map(f => f.properties[gradeProperty]).slice(0, 10);

console.log('Sample grades from data:', allGradesInData);

}

}

agriculturalLayersLoaded = true;

console.log('Agricultural layers loaded successfully');

} catch (error) {

console.error('Failed to load Agricultural layers:', error);

showToast('Failed to load Agricultural data', 'error');

}

}

function updateADPlantsControls() {

const container = document.querySelector('[data-group="ad-plants"]').nextElementSibling;

const adPlantGroups = [...new Set(adPlantsData.map(plant => plant.Group).filter(Boolean))];

container.innerHTML = adPlantGroups.map(group => {

const layerId = generateLayerId(group);

const color = getADPlantColor(group);

return `

<div class="layer-item flex items-center space-x-3 p-2 hover:bg-slate-50 rounded-lg">

<input type="checkbox" id="${layerId}" class="layer-checkbox" data-layer="${layerId}">

<div class="w-4 h-4 rounded-full" style="background-color: ${color}"></div>

<label for="${layerId}" class="text-sm text-slate-700 flex-1 cursor-pointer">${group}</label>

</div>

`;

}).join('');

}

function updateManureControls() {

console.log('UpdateManureControls called - checking for container');

const container = document.querySelector('[data-group="manure"]');

if (!container) {

console.warn('Manure button container not found');

return;

}

const contentContainer = container.nextElementSibling;

if (!contentContainer) {

console.warn('Manure content container not found');

return;

}

// Use the manure field mappings (8 categories) instead of MANURE\_LAYER\_CONFIGS

const categories = Object.keys(manureFieldMappings);

console.log('Found manure containers, updating with field mappings categories:', categories);

if (categories.length === 0) {

contentContainer.innerHTML = '<div class="p-4 text-sm text-slate-500">Loading manure categories...</div>';

return;

}

contentContainer.innerHTML = categories.map(category => {

const categoryId = category.toLowerCase().replace(/\s+/g, '-');

const color = manureVolumesColors[category] || '#808080';

return `

<div class="layer-item flex items-center space-x-3 p-2 hover:bg-slate-50 rounded-lg">

<input type="checkbox" id="manure-${categoryId}" class="layer-checkbox manure-checkbox" data-layer="manure-${categoryId}" onchange="updateManureDisplay()">

<div class="w-4 h-4 rounded" style="background-color: ${color}"></div>

<label for="manure-${categoryId}" class="text-sm text-slate-700 flex-1 cursor-pointer">${category}</label>

</div>

`;

}).join('');

console.log('Manure controls HTML updated successfully with', categories.length, 'categories');

}

// ==================== UTILITY FUNCTIONS ====================

// Aggregate manure volumes for a specific category from feature properties

function aggregateManureVolumes(properties, category) {

const fields = manureFieldMappings[category] || [];

let total = 0;

const details = {};

fields.forEach(fieldName => {

const value = properties[fieldName];

if (value && !isNaN(value)) {

total += parseFloat(value);

details[fieldName] = parseFloat(value);

}

});

return {

total: total,

details: details,

category: category

};

}

// Get color intensity for manure visualization using statistical normal distribution

function getManureIntensityColor(volume, category, categoryValues) {

const color = manureVolumesColors[category] || '#808080';

// Convert hex to RGB

const hex = color.replace('#', '');

let r = parseInt(hex.substr(0, 2), 16);

let g = parseInt(hex.substr(2, 2), 16);

let b = parseInt(hex.substr(4, 2), 16);

let opacity = 0.1;

if (categoryValues.length > 0 && volume > 0) {

// Calculate mean and standard deviation

const mean = categoryValues.reduce((sum, val) => sum + val, 0) / categoryValues.length;

const variance = categoryValues.reduce((sum, val) => sum + Math.pow(val - mean, 2), 0) / categoryValues.length;

const stdDev = Math.sqrt(variance);

// Calculate z-score (how many standard deviations from mean)

const zScore = (volume - mean) / stdDev;

// Map z-score to statistical percentiles using normal distribution

// Using approximate percentile boundaries for normal distribution

if (zScore >= 1.28) {

// Top 10% (z-score >= 1.28)

opacity = 0.85 + Math.min((zScore - 1.28) \* 0.1, 0.15); // 0.85 to 1.0

// Enhance coral/red tones for top values

if (category.includes('Beef') || category.includes('Dairy')) {

r = Math.min(255, r \* 1.2);

g = Math.max(0, g \* 0.7);

b = Math.max(0, b \* 0.7);

}

} else if (zScore >= 0.84) {

// 80-90th percentile (z-score 0.84 to 1.28)

opacity = 0.7 + ((zScore - 0.84) \* 0.34); // 0.7 to 0.85

} else if (zScore >= 0.52) {

// 70-80th percentile (z-score 0.52 to 0.84)

opacity = 0.55 + ((zScore - 0.52) \* 0.47); // 0.55 to 0.7

} else if (zScore >= 0.25) {

// 60-70th percentile (z-score 0.25 to 0.52)

opacity = 0.45 + ((zScore - 0.25) \* 0.37); // 0.45 to 0.55

} else if (zScore >= -0.25) {

// 40-60th percentile (z-score -0.25 to 0.25) - middle 20%

opacity = 0.3 + ((zScore + 0.25) \* 0.3); // 0.3 to 0.45

} else if (zScore >= -0.84) {

// 20-40th percentile (z-score -0.84 to -0.25)

opacity = 0.2 + ((zScore + 0.84) \* 0.17); // 0.2 to 0.3

} else {

// Bottom 20% (z-score < -0.84)

opacity = 0.1 + Math.max((zScore + 2) \* 0.05, 0); // 0.1 to 0.2

}

}

return {

r: Math.round(r),

g: Math.round(g),

b: Math.round(b),

opacity: Math.min(Math.max(opacity, 0.1), 1.0)

};

}

// Auto-update manure field mappings based on actual data

function updateManureFieldMappings(allFields) {

Object.keys(manureFieldMappings).forEach(category => {

manureFieldMappings[category] = [];

});

allFields.forEach(field => {

const lower = field.toLowerCase();

if (lower.includes('b\_fym') || (lower.includes('b\_') && lower.includes('fym'))) {

manureFieldMappings['Beef FYM'].push(field);

}

else if (lower.includes('b\_slu') || (lower.includes('b\_') && lower.includes('slu'))) {

manureFieldMappings['Beef Slurry'].push(field);

}

else if (lower === 'b\_grazing') {

manureFieldMappings['Beef FYM'].push(field);

}

else if (lower.includes('d\_fym') || (lower.includes('d\_') && lower.includes('fym'))) {

manureFieldMappings['Dairy FYM'].push(field);

}

else if (lower.includes('d\_slu') || (lower.includes('d\_') && lower.includes('slu'))) {

manureFieldMappings['Dairy Slurry'].push(field);

}

else if (lower === 'd\_grazing') {

manureFieldMappings['Dairy FYM'].push(field);

}

else if (lower.includes('bro\_fym') || lower.includes('bro\_') || (lower.includes('bro') && lower.includes('fym'))) {

manureFieldMappings['Broilers'].push(field);

}

else if (lower.includes('l\_fym') || lower.includes('l\_frrng') || (lower.includes('\_l\_') && lower.includes('fym'))) {

manureFieldMappings['Layers'].push(field);

}

else if (lower.includes('p\_fym') || lower.includes('p\_slu') || lower.includes('p\_outgrazi') ||

(lower.includes('\_p\_') && (lower.includes('fym') || lower.includes('slu') || lower.includes('out')))) {

manureFieldMappings['Pigs'].push(field);

}

else if (lower.includes('s\_fym') || lower.includes('s\_graz') ||

(lower.includes('\_s\_') && (lower.includes('fym') || lower.includes('graz')))) {

manureFieldMappings['Sheep'].push(field);

}

});

console.log('Updated manure field mappings:', manureFieldMappings);

}

function generateLayerId(group) {

return `ad-${group.toLowerCase()

.replace(/\s+/g, '-') // spaces to hyphens

.replace(/[&+]/g, '') // remove & and + symbols

.replace(/[/:]/g, '-') // forward slash and colon to hyphen

.replace(/[^a-z0-9-]/g, '') // remove any other special chars

.replace(/-+/g, '-') // collapse multiple hyphens

.replace(/^-|-$/g, '')}`; // remove leading/trailing hyphens

}

function formatNumber(num) {

if (num === 0) return '0';

if (!num || isNaN(num)) return 'N/A';

return num.toString().replace(/\B(?=(\d{3})+(?!\d))/g, ',');

}

// ==================== MAP UTILITY FUNCTIONS ====================

function createADPlantsGeoJSON(plants) {

const features = plants

.filter(plant => plant.Latitude && plant.Longitude)

.map(plant => ({

type: 'Feature',

properties: plant,

geometry: {

type: 'Point',

coordinates: [plant.Longitude, plant.Latitude]

}

}));

return {

type: 'FeatureCollection',

features

};

}

function createManureHeatMapGeoJSON(ladBoundaries, manureData, manureField) {

const features = ladBoundaries.features.map(feature => {

const ladCode = feature.properties.LAD23CD;

const manureRow = manureData.find(row => row['LAD code'] === ladCode);

const manureValue = manureRow ? (manureRow[manureField] || 0) : 0;

return {

...feature,

properties: {

...feature.properties,

manureValue: Number(manureValue),

manureData: manureRow

}

};

});

return {

type: 'FeatureCollection',

features

};

}

function getADPlantCircleRadius(capacity) {

if (capacity <= 25) return 4;

if (capacity <= 60) return 7;

return 10;

}

function getADPlantColor(group) {

return AD\_GROUP\_COLORS[group] || AD\_GROUP\_COLORS.Other;

}

function createHeatMapPaint(color, maxValue) {

// Ensure values are in ascending order and handle zero/negative max values

const safeMaxValue = Math.max(maxValue, 1);

const colorStops = [

0, `${color}00`,

safeMaxValue \* 0.2, `${color}33`,

safeMaxValue \* 0.4, `${color}66`,

safeMaxValue \* 0.6, `${color}99`,

safeMaxValue \* 0.8, `${color}CC`,

safeMaxValue, color

];

return {

'fill-color': [

'interpolate',

['linear'],

['get', 'manureValue'],

...colorStops

],

'fill-opacity': 0.7

};

}

function calculateManureTotals(manureData) {

const manureFields = ['Beef FYM', 'Dairy FYM', 'Dairy Slurry', 'Pig FYM', 'Pig Slurry', 'Sheep FYM', 'Layer', 'Broiler'];

return manureFields.reduce((total, field) => {

const value = manureData[field];

return total + (typeof value === 'number' ? value : 0);

}, 0);

}

// ==================== UI UPDATE FUNCTIONS ====================

function updatePlantsOverview() {

const container = document.getElementById('plants-overview');

if (!adPlantsData.length) {

container.innerHTML = '<div class="text-sm text-slate-500">No data loaded</div>';

return;

}

// Group plants by Group column and count them

const groupCounts = {};

adPlantsData.forEach(plant => {

const group = plant.Group || 'Other';

groupCounts[group] = (groupCounts[group] || 0) + 1;

});

// Sort groups by count (largest first)

const sortedGroups = Object.entries(groupCounts)

.sort(([,a], [,b]) => b - a);

container.innerHTML = `

<div class="text-sm text-slate-600 mb-3">

Total: <span class="font-semibold">${adPlantsData.length}</span> plants

</div>

${sortedGroups.map(([group, count]) => {

const color = getADPlantColor(group);

const layerId = generateLayerId(group);

const isActive = activeLayers.has(layerId);

return `

<div class="flex items-center justify-between p-2 hover:bg-slate-50 rounded-lg cursor-pointer transition-colors" onclick="toggleADLayer('${layerId}')">

<div class="flex items-center">

<div class="w-3 h-3 rounded-full mr-2" style="background-color: ${color}"></div>

<span class="text-sm text-slate-700">${group}</span>

</div>

<div class="flex items-center">

<span class="text-sm font-medium text-slate-900">${count}</span>

</div>

</div>

`;

}).join('')}

`;

}

async function toggleADLayer(layerId) {

console.log('Toggling AD layer:', layerId);

console.log('Available layers in registry:', Object.keys(layerManager.layers || {}));

// Auto-load Agricultural layers if attempting to toggle ALC grades

if (layerId.startsWith('alc-grade-') && !agriculturalLayersLoaded) {

console.log('Loading Agricultural layers for ALC grade toggle...');

await loadAgriculturalLayers();

}

// Auto-load Environmental layers on demand when toggling

if (layerId === 'aonb' && !layerManager.hasLayer('aonb')) {

console.log('Loading AONB layer on demand...');

await loadAONBLayer();

} else if (layerId === 'sssi' && !layerManager.hasLayer('sssi')) {

console.log('Loading SSSI layer on demand...');

await loadSSSILayer();

} else if (layerId === 'nvz' && !layerManager.hasLayer('nvz')) {

console.log('Loading NVZ layer on demand...');

await loadNVZLayer();

} else if (layerId === 'flood' && !layerManager.hasLayer('flood')) {

console.log('Loading Flood layer on demand...');

await loadFloodLayer();

}

// Ensure environmental section is initialized

if (['aonb', 'sssi', 'nvz', 'flood'].includes(layerId) && !environmentalLayersLoaded) {

console.log('Initializing environmental layers section...');

await loadEnvironmentalLayers();

}

console.log('Active layers:', Array.from(activeLayers));

const isCurrentlyActive = activeLayers.has(layerId);

if (isCurrentlyActive) {

activeLayers.delete(layerId);

if (layerManager && layerManager.layers[layerId]) {

layerManager.hide(layerId);

console.log('Hiding layer:', layerId);

} else {

console.warn('Layer not found in registry:', layerId);

}

} else {

activeLayers.add(layerId);

if (layerManager && layerManager.layers[layerId]) {

layerManager.show(layerId);

console.log('Showing layer:', layerId);

} else {

console.warn('Layer not found in registry:', layerId);

}

}

// Update the plants overview to reflect current state

updatePlantsOverview();

updateLegend();

}

function updateLayerControls() {

const container = document.getElementById('layer-controls');

let html = '';

// AD Plants Group - show loading initially

html += `

<div class="border-b border-slate-200">

<button class="accordion-trigger w-full p-4 text-left hover:bg-slate-50 transition-colors" data-group="ad-plants">

<div class="flex items-center justify-between">

<div class="flex items-center">

<span class="text-lg mr-3">🏭</span>

<span class="font-semibold text-slate-900">AD Plants</span>

</div>

<svg class="accordion-icon h-4 w-4 text-slate-500 transition-transform" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 9l-7 7-7-7"></path>

</svg>

</div>

</button>

<div class="accordion-content collapsed space-y-2">

<div class="text-sm text-slate-500">Click to load AD plants data</div>

</div>

</div>

`;

// Manure Volumes Group - show loading initially

html += `

<div class="border-b border-slate-200">

<button class="accordion-trigger w-full p-4 text-left hover:bg-slate-50 transition-colors" data-group="manure">

<div class="flex items-center justify-between">

<div class="flex items-center">

<span class="text-lg mr-3">📊</span>

<span class="font-semibold text-slate-900">Manure Volumes by Livestock</span>

</div>

<svg class="accordion-icon h-4 w-4 text-slate-500 transition-transform" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 9l-7 7-7-7"></path>

</svg>

</div>

</button>

<div class="accordion-content collapsed space-y-2">

<div class="text-sm text-slate-500">Click to load manure data layers</div>

</div>

</div>

`;

// Other Layer Groups (skip just the manure group at index 0)

const otherGroups = LAYER\_GROUPS.slice(1); // Skip only the manure group

otherGroups.forEach(group => {

const iconColors = {

boundaries: 'text-slate-600',

environmental: 'text-green-600',

agricultural: 'text-amber-600',

roads: 'text-slate-600',

infrastructure: 'text-blue-600'

};

html += `

<div class="border-b border-slate-200">

<button class="accordion-trigger w-full p-4 text-left hover:bg-slate-50 transition-colors" data-group="${group.id}">

<div class="flex items-center justify-between">

<div class="flex items-center">

<span class="text-lg mr-3">${group.icon}</span>

<span class="font-semibold text-slate-900">${group.name}</span>

</div>

<svg class="accordion-icon h-4 w-4 text-slate-500 transition-transform" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 9l-7 7-7-7"></path>

</svg>

</div>

</button>

<div class="accordion-content collapsed space-y-2">

${group.layers.map(layerId => {

const layerNames = {

lad: 'Local Authority Districts',

lpa: 'Local Planning Authorities',

aonb: 'AONB',

sssi: 'SSSI',

nvz: 'Nitrate Vulnerable Zones',

flood: 'Historic Flood Areas',

'alc-grade-1': 'Grade 1 (Best Quality)',

'alc-grade-2': 'Grade 2 (Very Good)',

'alc-grade-3': 'Grade 3 (Good to Moderate)',

'alc-grade-4': 'Grade 4 (Poor)',

'alc-grade-5': 'Grade 5 (Very Poor)',

roads: 'Major Roads',

dno: 'DNO Regions',

water: 'Water Companies',

nts: 'Gas Grid (NTS)',

brownfield: 'Brownfield Land'

};

const layerColors = {

lad: '#7e3aa2', // Purple for LAD boundaries

lpa: '#8b5a3c', // Brown for LPA boundaries

aonb: '#86efac',

sssi: '#34d399',

nvz: '#fca5a5',

flood: '#93c5fd',

'alc-grade-1': '#22c55e', // Green - Best quality

'alc-grade-2': '#84cc16', // Lime green

'alc-grade-3': '#eab308', // Yellow

'alc-grade-4': '#f97316', // Orange

'alc-grade-5': '#ef4444', // Red - Poorest quality

'brownfield': '#92400e', // Brown for brownfield land

roads: '#64748b',

dno: '#3b82f6',

water: '#60a5fa',

nts: '#f97316',

population: '#a855f7'

};

const isChecked = layerId === 'lad' ? 'checked' : '';

return `

<div class="layer-item flex items-center space-x-3 p-2 hover:bg-slate-50 rounded-lg">

<input type="checkbox" id="${layerId}" class="layer-checkbox" data-layer="${layerId}" ${isChecked}>

<div class="w-4 h-4 ${layerId === 'roads' || layerId === 'nts' ? 'h-1' : 'rounded'}" style="background-color: ${layerColors[layerId]}"></div>

<label for="${layerId}" class="text-sm text-slate-700 flex-1 cursor-pointer">${layerNames[layerId] || layerId.toUpperCase()}</label>

</div>

`;

}).join('')}

</div>

</div>

`;

});

container.innerHTML = html;

}

function updateLegend() {

const legendContainer = document.getElementById('map-legend');

const legendContent = document.getElementById('legend-content');

const activeLayersList = Array.from(activeLayers);

if (activeLayersList.length === 0 || (activeLayersList.length === 1 && activeLayersList[0] === 'lad')) {

legendContainer.classList.add('hidden');

return;

}

legendContainer.classList.remove('hidden');

let html = '';

// AD Plants legend

const visibleADLayers = activeLayersList.filter(layer => layer.startsWith('ad-'));

if (visibleADLayers.length > 0) {

html += '<div class="mb-4"><div class="font-medium text-slate-900 mb-2">AD Plants</div><div class="space-y-1">';

Object.entries(AD\_GROUP\_COLORS).forEach(([group, color]) => {

html += `

<div class="flex items-center text-xs text-slate-600">

<div class="w-3 h-3 rounded-full mr-2" style="background-color: ${color}"></div>

<span>${group}</span>

</div>

`;

});

html += '</div></div>';

}

// Manure heat maps legend

const visibleManureLayers = activeLayersList.filter(layer => layer.startsWith('manure-'));

visibleManureLayers.forEach(layerId => {

const config = MANURE\_LAYER\_CONFIGS.find(c => c.id === layerId);

if (config) {

html += `

<div class="mb-4">

<div class="font-medium text-slate-900 mb-2">${config.name} Manure</div>

<div class="flex items-center space-x-1">

<span class="text-xs text-slate-600">Low</span>

<div class="flex-1 h-3 rounded" style="background: linear-gradient(to right, ${config.color}20, ${config.color}40, ${config.color}60, ${config.color}80, ${config.color})"></div>

<span class="text-xs text-slate-600">High</span>

</div>

</div>

`;

}

});

// Other layers legend

const otherLayers = activeLayersList.filter(layer =>

!layer.startsWith('ad-') && !layer.startsWith('manure-') && layer !== 'lad'

);

if (otherLayers.length > 0) {

html += '<div><div class="font-medium text-slate-900 mb-2">Other Layers</div><div class="space-y-1">';

otherLayers.forEach(layer => {

html += `

<div class="flex items-center text-xs text-slate-600">

<div class="w-3 h-1 mr-2 bg-slate-500"></div>

<span>${layer.toUpperCase().replace('-', ' ')}</span>

</div>

`;

});

html += '</div></div>';

}

legendContent.innerHTML = html;

}

// ==================== INFO PANEL FUNCTIONS ====================

function openInfoPanel(data) {

console.log('Opening info panel with data:', data);

currentInfoPanelData = data;

const panel = document.getElementById('info-panel');

const title = document.getElementById('info-panel-title');

const content = document.getElementById('info-panel-content');

if (!panel || !title || !content) {

console.error('Info panel elements not found:', {panel: !!panel, title: !!title, content: !!content});

return;

}

if (data.type === 'plant') {

title.textContent = data.plant['Site Name'] || 'AD Plant Details';

content.innerHTML = renderPlantInfo(data.plant);

} else if (data.type === 'lad') {

// Use the comprehensive LAD handler instead of simple renderLADInfo

displayLADInfoPanel({

ladProperties: data.ladProperties,

plantsInLAD: data.plantsInLAD || [],

manureData: data.manureData

});

return; // Early return since displayLADInfoPanel handles the panel opening

} else if (data.type === 'lpa') {

// Use the comprehensive LPA handler similar to LAD

displayLPAInfoPanel({

lpaProperties: data.lpaProperties,

plantsInLPA: data.plantsInLPA || [],

manureData: data.manureInLPA

});

return; // Early return since displayLPAInfoPanel handles the panel opening

}

panel.classList.remove('translate-x-full');

panel.classList.add('translate-x-0');

console.log('Info panel should now be visible');

}

function closeInfoPanel() {

const panel = document.getElementById('info-panel');

panel.classList.add('translate-x-full');

panel.classList.remove('open');

currentInfoPanelData = null;

}

function renderPlantInfo(plant) {

const getStatusColor = (group) => {

const colors = {

'Operational: GtG + GtG&CHP': '#0f5132',

'Operational: CHP': '#198754',

'Operational: Other': '#6f9c6f',

'Under Construction': '#2574ce',

'Planning Granted': '#854ec2',

'Planning Application': '#f38c1d',

'Planning Submitted': '#f38c1d',

'Proposed': '#f38c1d',

'Refused': '#b6222b',

'Withdrawn': '#b6222b'

};

return colors[group] || '#6b7280';

};

const statusColor = getStatusColor(plant.Group || plant.Status);

const capacity = parseFloat(String(plant['Capacity(kWe)'] || 0).replace(/,/g, ''));

const totalFeedstock = parseFloat(String(plant['Totalfeedstock(tpa)'] || 0).replace(/,/g, ''));

const manure = parseFloat(String(plant['Manure/Slurry(tpa)'] || 0).replace(/,/g, ''));

const crop = parseFloat(String(plant['Crop(tpa)'] || 0).replace(/,/g, ''));

const foodWaste = parseFloat(String(plant['FoodWaste(tpa)'] || 0).replace(/,/g, ''));

const cropWaste = parseFloat(String(plant['CropWaste(tpa)'] || 0).replace(/,/g, ''));

const otherWaste = parseFloat(String(plant['OtherWaste(tpa)'] || 0).replace(/,/g, ''));

return `

<div class="space-y-4">

<!-- Header Section -->

<div class="space-y-3">

<div class="px-3 py-1 rounded-full text-sm font-medium text-white" style="background-color: ${statusColor}">

${plant.Group || plant.Status || 'Unknown Status'}

</div>

<div>

<h2 class="text-xl font-bold text-slate-900 mb-1">${plant['Site name'] || 'Anaerobic Digestion Plant'}</h2>

<p class="text-slate-600 flex items-center">

📍 ${plant.Postcode || 'Location not specified'}

</p>

</div>

</div>

<!-- Project Details Section -->

<div class="bg-slate-50 rounded-lg p-4">

<h4 class="font-semibold text-slate-900 mb-3 flex items-center">

🏗️ Project Details

</h4>

<div class="grid grid-cols-1 gap-2 text-sm">

<div class="flex justify-between">

<span class="text-slate-600">Developer:</span>

<span class="font-medium text-slate-900">${plant.Developer || 'N/A'}</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">Completion:</span>

<span class="font-medium text-slate-900">${plant.Completion || 'N/A'}</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">Plant Type:</span>

<span class="font-medium text-slate-900">${plant.Output || plant.Type || 'N/A'}</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">Postcode:</span>

<span class="font-medium text-slate-900">${plant.Postcode || 'N/A'}</span>

</div>

</div>

</div>

<!-- Capacity Section -->

<div class="grid grid-cols-2 gap-3">

<div class="bg-white border-2 rounded-lg p-4 text-center" style="border-color: ${statusColor}">

<h5 class="text-sm font-medium text-slate-600 mb-1">Electrical Capacity</h5>

<div class="text-2xl font-bold text-blue-600">${formatNumber(capacity)}</div>

<div class="text-sm text-slate-500">kWe</div>

</div>

<div class="bg-white border-2 rounded-lg p-4 text-center" style="border-color: ${statusColor}">

<h5 class="text-sm font-medium text-slate-600 mb-1">Energy Output</h5>

<div class="text-2xl font-bold text-green-600">${plant.GWh || 'N/A'}</div>

<div class="text-sm text-slate-500">GWh/year</div>

</div>

</div>

<!-- Feedstock Section -->

<div class="bg-white rounded-lg border-2 p-4" style="border-color: ${statusColor}">

<h4 class="font-semibold text-slate-900 mb-3">Feedstock</h4>

<div class="space-y-3">

<div>

<div class="text-sm text-slate-600 mb-1">Feedstock Type:</div>

<div class="text-sm font-medium text-slate-900">${plant.Feedstock || 'N/A'}</div>

</div>

<div>

<div class="text-sm text-slate-600 mb-1">Total Feedstock:</div>

<div class="text-lg font-bold text-slate-900">${formatNumber(totalFeedstock)} tpa</div>

</div>

<div class="border-t pt-3">

<div class="text-sm font-medium text-slate-700 mb-2">Detailed Breakdown:</div>

<div class="space-y-1 text-sm">

<div class="flex justify-between items-center">

<span class="text-slate-600">🐄 Manure/Slurry:</span>

<span class="font-medium">${formatNumber(manure)} tpa</span>

</div>

<div class="flex justify-between items-center">

<span class="text-slate-600">🌽 Crop:</span>

<span class="font-medium">${formatNumber(crop)} tpa</span>

</div>

<div class="flex justify-between items-center">

<span class="text-slate-600">🍎 Food Waste:</span>

<span class="font-medium">${formatNumber(foodWaste)} tpa</span>

</div>

<div class="flex justify-between items-center">

<span class="text-slate-600">🌿 Crop Waste:</span>

<span class="font-medium">${formatNumber(cropWaste)} tpa</span>

</div>

<div class="flex justify-between items-center">

<span class="text-slate-600">♻️ Other Waste:</span>

<span class="font-medium">${formatNumber(otherWaste)} tpa</span>

</div>

</div>

</div>

</div>

</div>

<!-- Planning and Utilities Section -->

<div class="bg-white rounded-lg border-2 p-4" style="border-color: ${statusColor}">

<h4 class="font-semibold text-slate-900 mb-3 flex items-center">

📋 Planning and Utilities

</h4>

<div class="space-y-2 text-sm">

${plant['Planning Application Reference'] && plant['Planning Application Reference'].trim() ? `

<div class="flex justify-between items-center">

<span class="text-slate-600">Planning Reference:</span>

<a href="https://www.google.com/search?q=${encodeURIComponent(plant['Planning Application Reference'] + ' planning application')}"

target="\_blank" rel="noopener noreferrer"

class="text-blue-600 hover:text-blue-800 font-medium">

${plant['Planning Application Reference']}

</a>

</div>

` : ''}

<div class="flex justify-between">

<span class="text-slate-600">Water Company:</span>

<span class="font-medium text-slate-900">${plant.WaterCo || 'Unknown'}</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">DNO:</span>

<span class="font-medium text-slate-900">${plant.DNO || 'Unknown'}</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">Planning Authority:</span>

<span class="font-medium text-slate-900">${plant['lpa\_name'] || 'N/A'}</span>

</div>

</div>

</div>

</div>

`;

}

function renderLADInfo(properties, manureData) {

const ladCode = properties.LAD23CD;

const ladName = properties.LAD23NM;

// Find AD plants in this LAD (using correct column name)

const plantsInLAD = adPlantsData.filter(plant => plant['lad\_name'] === ladName);

// Define status colors

const statusColors = {

'Operational': '#0f5132',

'Under Construction': '#2574ce',

'Planning Granted': '#854ec2',

'Planning Application': '#f38c1d',

'Planning Submitted': '#f38c1d',

'Proposed': '#f38c1d',

'Refused': '#b6222b',

'Withdrawn': '#b6222b'

};

// Group plants by status

const plantsByStatus = {};

plantsInLAD.forEach(plant => {

const status = plant.Status || 'Unknown';

if (!plantsByStatus[status]) {

plantsByStatus[status] = [];

}

plantsByStatus[status].push(plant);

});

// Calculate crop analysis (placeholder for future enhancement)

const totalLADArea = 100; // km²

const cropCoveragePercent = 25;

const totalEnergyPotential = 150; // GWh

return `

<div class="space-y-6">

<!-- Header Section -->

<div class="bg-white rounded-lg shadow-sm border border-gray-200 p-6" style="border-radius: 8px;">

<h2 style="font-size: 1.4em; font-weight: 700; color: #2c3e50; margin-bottom: 0;">

📍 ${ladName}

</h2>

</div>

<!-- AD Plants Section -->

<div class="bg-white rounded-lg shadow-sm border border-gray-200 p-6" style="border-radius: 8px;">

<h3 style="font-weight: 600; color: #2c3e50; margin-bottom: 16px; font-size: 1.1em;">

AD Plants in ${ladName}

</h3>

${Object.keys(plantsByStatus).length > 0 ? `

${Object.entries(plantsByStatus).map(([status, plants]) => `

<details class="mb-4" style="border: 1px solid #e9ecef; border-radius: 8px; overflow: hidden;">

<summary style="

background: linear-gradient(135deg, ${statusColors[status] || '#6c757d'}, ${statusColors[status] || '#6c757d'}dd);

color: white;

padding: 12px 16px;

font-weight: 600;

cursor: pointer;

border: none;

outline: none;

">

${status} (${plants.length})

</summary>

<div style="padding: 16px;">

${plants.map(plant => `

<div style="

border-left: 4px solid ${statusColors[status] || '#6c757d'};

background: #fafafa;

margin-bottom: 12px;

padding: 16px;

border-radius: 0 8px 8px 0;

box-shadow: 0 2px 4px rgba(0,0,0,0.05);

">

<div style="font-weight: 700; color: #2c3e50; margin-bottom: 8px;">

${plant['Site name'] || 'Unnamed Plant'}

</div>

<div style="color: #6c757d; font-size: 0.9em; margin-bottom: 8px;">

📍 ${plant['lpa\_name'] || 'Unknown LPA'} • ${plant['Postcode'] || 'No postcode'}<br>

🌾 ${plant['Feedstock'] || 'Mixed feedstock'}

</div>

<div style="display: grid; grid-template-columns: 1fr 1fr; gap: 8px; font-size: 0.85em;">

<div>

<strong>Feedstock:</strong> ${formatNumber(parseFloat(plant['Totalfeedstock(tpa)'] || 0))} tpa

</div>

<div>

<strong>Capacity:</strong> ${formatNumber(parseFloat(plant['Capacity(kWe)'] || 0))} kWe

</div>

${plant['Biomethanecapacity(Nm3/hrbiomethane)'] ? `

<div style="grid-column: span 2;">

<strong>Biomethane:</strong> ${formatNumber(parseFloat(plant['Biomethanecapacity(Nm3/hrbiomethane)']))} Nm³/hr

</div>

` : ''}

</div>

</div>

`).join('')}

</div>

</details>

`).join('')}

` : `

<div style="text-align: center; padding: 32px; color: #6c757d;">

<div style="font-size: 2em; margin-bottom: 16px;">🏭</div>

<div>No AD plants found in this LAD</div>

</div>

`}

</div>

<!-- Crop Analysis Section -->

<div class="bg-white rounded-lg shadow-sm border border-gray-200 p-6" style="border-radius: 8px;">

<details>

<summary style="

background: linear-gradient(135deg, #28a745, #28a745dd);

color: white;

padding: 12px 16px;

font-weight: 600;

cursor: pointer;

border: none;

outline: none;

border-radius: 8px;

margin: -6px -6px 16px -6px;

">

🌾 Crop Analysis for ${ladName}

</summary>

<div style="padding-top: 16px;">

<div style="display: grid; grid-template-columns: 1fr 1fr; gap: 16px; margin-bottom: 16px;">

<div>

<strong>Total LAD Area:</strong> ~${totalLADArea} km²

</div>

<div>

<strong>Crop Coverage:</strong> ~${cropCoveragePercent}%

</div>

</div>

<div style="margin-bottom: 16px;">

<h4 style="font-weight: 600; color: #2c3e50; margin-bottom: 12px;">

Top Crops by Biomethane Potential

</h4>

<div style="space-y: 8px;">

<div style="display: flex; justify-content: space-between; padding: 8px; background: #f8f9fa; border-radius: 4px; margin-bottom: 4px;">

<span>🌽 Maize</span>

<span style="color: #28a745; font-weight: 600;">${Math.round(totalEnergyPotential \* 0.4)} GWh</span>

</div>

<div style="display: flex; justify-content: space-between; padding: 8px; background: #f8f9fa; border-radius: 4px; margin-bottom: 4px;">

<span>🌾 Wheat</span>

<span style="color: #28a745; font-weight: 600;">${Math.round(totalEnergyPotential \* 0.3)} GWh</span>

</div>

<div style="display: flex; justify-content: space-between; padding: 8px; background: #f8f9fa; border-radius: 4px; margin-bottom: 4px;">

<span>🥔 Root Crops</span>

<span style="color: #28a745; font-weight: 600;">${Math.round(totalEnergyPotential \* 0.2)} GWh</span>

</div>

<div style="display: flex; justify-content: space-between; padding: 8px; background: #f8f9fa; border-radius: 4px;">

<span>🌿 Other</span>

<span style="color: #28a745; font-weight: 600;">${Math.round(totalEnergyPotential \* 0.1)} GWh</span>

</div>

</div>

</div>

<div style="

border-top: 2px solid #28a745;

padding-top: 16px;

text-align: center;

font-weight: 700;

color: #2c3e50;

">

Total Energy Potential: ${totalEnergyPotential} GWh

</div>

</div>

</details>

</div>

${manureData ? `

<!-- Livestock Resources -->

<div class="bg-white rounded-lg shadow-sm border border-gray-200 p-6" style="border-radius: 8px;">

<h4 style="font-weight: 600; color: #2c3e50; margin-bottom: 16px;">Livestock Manure Resources</h4>

<div style="color: #6c757d; font-size: 0.9em; margin-bottom: 12px;">Available feedstock (tonnes/year)</div>

<div style="display: grid; gap: 8px;">

${MANURE\_LAYER\_CONFIGS.map(config => {

const value = manureData[config.field] || 0;

if (value === 0) return '';

return `

<div style="display: flex; justify-content: space-between; align-items: center; padding: 8px; background: #f8f9fa; border-radius: 4px;">

<div style="display: flex; align-items: center;">

<div style="width: 12px; height: 12px; border-radius: 50%; background-color: ${config.color}; margin-right: 8px;"></div>

<span>${config.name}</span>

</div>

<span style="font-weight: 600;">${formatNumber(value)}</span>

</div>

`;

}).filter(html => html).join('')}

</div>

</div>

` : ''}

</div>

`;

}

// LPA Info Panel Display - copy of LAD functionality

function displayLPAInfoPanel(data) {

console.log('Displaying LPA info panel');

const panel = document.getElementById('info-panel');

const title = document.getElementById('info-panel-title');

const content = document.getElementById('info-panel-content');

if (!panel || !title || !content) {

console.error('Info panel elements not found');

return;

}

const lpaName = data.lpaProperties.LPA22NM || 'Unknown LPA';

const lpaCode = data.lpaProperties.LAD22CD;

console.log(`Setting up info panel for LPA: ${lpaName}`);

title.textContent = `📍 ${lpaName}`;

// Group plants by group (which matches the AD layer groups) - same as LAD

const statusGroups = {};

data.plantsInLPA.forEach(plant => {

const group = plant.Group || 'Unknown';

if (!statusGroups[group]) {

statusGroups[group] = [];

}

statusGroups[group].push(plant);

});

// Group colors matching AD plant layer colors - same as LAD

const statusColors = {

'Operational: GtG + GtG&CHP': '#0f5132',

'Operational: CHP': '#198754',

'Operational: Other': '#6f9c6f',

'Planning Submitted': '#f38c1d',

'Planning Refused / Withdrawn / Abandoned': '#6b7280',

'Planning Granted': '#854ec2',

'Under Construction': '#2574ce',

'Operational': '#0f5132',

'Planning Application': '#f38c1d',

'Proposed': '#f38c1d',

'Refused': '#b6222b',

'Withdrawn': '#b6222b'

};

// Shortened status names for display - same as LAD

const statusDisplayNames = {

'Planning Refused / Withdrawn / Abandoned': 'Refused/Withdrawn/Abandoned'

};

// Calculate statistics - same as LAD

const totalPlants = data.plantsInLPA.length;

const operationalPlants = data.plantsInLPA.filter(p => p.Status && p.Status.includes('Operational')).length;

const totalCapacity = data.plantsInLPA.reduce((sum, plant) => {

const capacity = parseFloat(String(plant['Capacity(kWe)']).replace(/,/g, '')) || 0;

return sum + capacity;

}, 0);

// Calculate total GWh capacity for overview - same as LAD

const totalGWhCapacity = data.plantsInLPA.reduce((sum, plant) => {

const gwh = parseFloat(plant.GWh) || 0;

return sum + gwh;

}, 0);

let html = `

<div class="space-y-6">

<!-- Header Card - exactly same as LAD -->

<div class="bg-gradient-to-br from-blue-600 to-indigo-700 text-white border border-blue-800 p-6">

<div class="mb-4">

<h2 class="text-xl font-bold">${lpaName}</h2>

</div>

<!-- Key Metrics Grid - exactly same as LAD -->

<div class="grid grid-cols-2 gap-3">

<div class="bg-white bg-opacity-10 border border-white border-opacity-20 p-3 text-center h-16 flex flex-col justify-center">

<div class="text-lg font-bold">${totalPlants}</div>

<div class="text-xs opacity-90">Total Plants</div>

</div>

<div class="bg-white bg-opacity-10 border border-white border-opacity-20 p-3 text-center h-16 flex flex-col justify-center">

<div class="text-lg font-bold text-green-300">${operationalPlants}</div>

<div class="text-xs opacity-90">Operational</div>

</div>

<div class="bg-white bg-opacity-10 border border-white border-opacity-20 p-3 text-center h-16 flex flex-col justify-center">

<div class="text-lg font-bold text-yellow-300">${formatNumber(Math.round(totalCapacity / 1000))}</div>

<div class="text-xs opacity-90">MWe Capacity</div>

</div>

<div class="bg-white bg-opacity-10 border border-white border-opacity-20 p-3 text-center h-16 flex flex-col justify-center">

<div class="text-lg font-bold text-orange-300">${formatNumber(totalGWhCapacity)}</div>

<div class="text-xs opacity-90">GWh/year</div>

</div>

</div>

</div>

`;

// Plants by Status - Modern Card Design - exactly same as LAD

if (Object.keys(statusGroups).length > 0) {

html += `

<div class="bg-white border border-gray-200 overflow-hidden">

<div class="bg-gray-50 px-6 py-4 border-b border-gray-200">

<button class="w-full text-left flex items-center justify-between font-semibold text-gray-800 hover:text-blue-600 transition-colors" onclick="togglePlantsContent(this)">

<div class="flex items-center">

<div class="w-10 h-10 bg-blue-600 rounded-lg flex items-center justify-center mr-3">

<svg class="w-5 h-5 text-white" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 21V5a2 2 0 00-2-2H7a2 2 0 00-2 2v16m14 0h2m-2 0h-5m-9 0H3m2 0h5M9 7h1m-1 4h1m4-4h1m-1 4h1m-5 10v-5a1 1 0 011-1h2a1 1 0 011 1v5m-4 0h4"></path>

</svg>

</div>

<div>

<h3 class="text-lg font-semibold">AD Plants by Status</h3>

<p class="text-sm text-gray-600">${totalPlants} plants in this LPA</p>

</div>

</div>

<svg class="w-5 h-5 text-gray-400 transform transition-transform" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 9l-7 7-7-7"></path>

</svg>

</button>

</div>

<div class="plants-content" style="display: none;">

`;

Object.entries(statusGroups).forEach(([group, plants]) => {

const color = statusColors[group] || '#6b7280';

const displayName = statusDisplayNames[group] || group;

html += `

<div class="border-b border-gray-200 last:border-b-0">

<div class="px-6 py-4" style="background-color: ${color}20; border-left: 4px solid ${color};">

<button class="w-full text-left flex items-center justify-between" onclick="toggleStatusGroup('${group.replace(/[^a-zA-Z0-9]/g, '-')}')">

<div class="flex items-center">

<div class="w-3 h-3 rounded-full mr-3" style="background-color: ${color};"></div>

<h4 class="font-medium text-gray-900">${displayName}</h4>

<span class="ml-2 text-sm text-gray-600">(${plants.length})</span>

</div>

<svg class="w-4 h-4 text-gray-400" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 9l-7 7-7-7"></path>

</svg>

</button>

<div id="group-${group.replace(/[^a-zA-Z0-9]/g, '-')}" style="display: none;" class="mt-4 space-y-3">

${plants.map((plant, index) => {

const capacity = parseFloat(String(plant['Capacity(kWe)']).replace(/,/g, '')) || 0;

const siteName = plant['Site name'] || plant['Site Name'] || 'Unknown Site';

const location = plant.Address || plant.Postcode || 'Location not specified';

const completion = plant.Completion || 'Date not specified';

const feedstock = plant.Feedstock || 'Feedstock not specified';

const latitude = plant.Latitude;

const longitude = plant.Longitude;

return `

<div class="bg-white border border-gray-200 rounded-lg p-4 hover:shadow-sm transition-shadow">

<div class="flex justify-between items-start mb-2">

<h5 class="font-medium text-gray-900 text-sm">${siteName}</h5>

<div class="text-right">

<div class="text-sm font-medium text-blue-600">${formatNumber(capacity)} kWe</div>

${latitude && longitude ? `

<button onclick="flyToPlant(${latitude}, ${longitude}, '${siteName.replace(/'/g, "\\'")}');"

class="text-xs text-blue-500 hover:text-blue-700 underline mt-1">

View on Map

</button>

` : ''}

</div>

</div>

<div class="space-y-1 text-xs text-gray-600">

<div class="flex items-center">

<span class="w-4">📍</span>

<span>${location}</span>

</div>

<div class="flex items-center">

<span class="w-4">🏗️</span>

<span>Completed: ${completion}</span>

</div>

<div class="flex items-start">

<span class="w-4">🌿</span>

<span class="leading-relaxed">

<strong>Feedstock:</strong> ${feedstock}

</span>

</div>

</div>

</div>

`;

}).join('')}

</div>

</div>

</div>

`;

});

html += `

</div>

</div>

`;

} else {

html += `

<div class="bg-white border border-gray-200 overflow-hidden">

<div class="bg-gray-50 px-6 py-4 border-b border-gray-200">

<div class="flex items-center">

<div class="w-10 h-10 bg-gray-400 rounded-lg flex items-center justify-center mr-3">

<svg class="w-5 h-5 text-white" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M19 21V5a2 2 0 00-2-2H7a2 2 0 00-2 2v16m14 0h2m-2 0h-5m-9 0H3m2 0h5M9 7h1m-1 4h1m4-4h1m-1 4h1m-5 10v-5a1 1 0 011-1h2a1 1 0 011 1v5m-4 0h4"></path>

</svg>

</div>

<div>

<h3 class="text-lg font-semibold text-gray-500">No AD Plants Found</h3>

<p class="text-sm text-gray-400">This LPA area has no anaerobic digestion plants</p>

</div>

</div>

</div>

</div>

`;

}

// Manure Analysis - exactly same as LAD design

console.log('Looking for manure data for LPA:', lpaName, lpaCode);

console.log('Available LPA manure data entries:', window.lpaManureData ? window.lpaManureData.length : 'NO DATA');

const lpaManureEntry = findManureDataForLPA(lpaName, lpaCode);

console.log('Found LPA manure entry:', lpaManureEntry);

if (lpaManureEntry) {

// Handle both string and number values for totals

const totalTonnesValue = lpaManureEntry['total\_tonnes'];

const totalGwhValue = lpaManureEntry['total\_gwh'];

const totalTonnes = parseFloat(

typeof totalTonnesValue === 'string' ? totalTonnesValue.replace(/,/g, '') : totalTonnesValue || 0

);

const totalGwh = parseFloat(

typeof totalGwhValue === 'string' ? totalGwhValue.replace(/,/g, '') : totalGwhValue || 0

);

html += `

<div class="bg-white border border-gray-200 overflow-hidden">

<div class="bg-gradient-to-r from-green-50 to-amber-50 px-6 py-4 border-b border-gray-200">

<button class="w-full text-left flex items-center justify-between font-semibold text-gray-800 hover:text-green-600 transition-colors" onclick="toggleManureContent(this)">

<div class="flex items-center">

<div class="w-10 h-10 bg-green-600 rounded-lg flex items-center justify-center mr-3">

<svg class="w-5 h-5 text-white" fill="currentColor" viewBox="0 0 24 24">

<path d="M12.5 8.5v7h-1v-7h1zm-8 7V12a4 4 0 014-4h8a4 4 0 014 4v3.5h-1V12a3 3 0 00-3-3h-8a3 3 0 00-3 3v3.5h-1zm12-11.5a1.5 1.5 0 11-3 0 1.5 1.5 0 013 0zm1 0a2.5 2.5 0 10-5 0 2.5 2.5 0 005 0zm-2.5-1.5a.5.5 0 00-.5.5v1a.5.5 0 001 0V3a.5.5 0 00-.5-.5z"/>

<path d="M12 20c-4.4 0-8-3.6-8-8s3.6-8 8-8 8 3.6 8 8-3.6 8-8 8zm0-1c3.9 0 7-3.1 7-7s-3.1-7-7-7-7 3.1-7 7 3.1 7 7 7z"/>

<path d="M12 6c-1.1 0-2 .9-2 2v4c0 1.1.9 2 2 2s2-.9 2-2V8c0-1.1-.9-2-2-2z"/>

</svg>

</div>

<div>

<h3 class="text-lg font-semibold">Manure Production Analysis</h3>

<p class="text-sm text-gray-600">Annual livestock waste generation</p>

</div>

</div>

<div class="text-right">

<div class="text-sm font-medium text-green-700">${formatNumber(totalTonnes)} tonnes/year</div>

<div class="text-xs text-gray-600">${formatNumber(totalGwh)} GWh potential</div>

</div>

</button>

</div>

<div class="manure-content" style="display: none;">

<div class="px-6 py-4 bg-gradient-to-r from-green-50 to-amber-50">

<div class="grid grid-cols-3 gap-4 text-center">

<div class="bg-white bg-opacity-60 rounded-lg p-3">

<div class="text-lg font-bold text-green-700">${formatNumber(totalTonnes)}</div>

<div class="text-xs text-gray-600">Total Tonnes/Year</div>

</div>

<div class="bg-white bg-opacity-60 rounded-lg p-3">

<div class="text-lg font-bold text-amber-700">${formatNumber(totalGwh)}</div>

<div class="text-xs text-gray-600">GWh Potential</div>

</div>

<div class="bg-white bg-opacity-60 rounded-lg p-3">

<div class="text-lg font-bold text-blue-700">${formatNumber(Math.round(totalGwh / totalGwh \* 100) || 0)}%</div>

<div class="text-xs text-gray-600">Efficiency</div>

</div>

</div>

</div>

<div class="px-6 py-4">

<h4 class="font-semibold text-gray-800 mb-3">Breakdown by Livestock Type</h4>

<div class="space-y-2">

${generateManureBreakdown(lpaManureEntry)}

</div>

</div>

</div>

</div>

`;

} else {

console.log('No manure data found for this LPA');

if (window.lpaManureData && window.lpaManureData.length > 0) {

html += `

<div class="bg-white border border-gray-200 overflow-hidden">

<div class="bg-gradient-to-r from-green-50 to-amber-50 px-6 py-4 border-b border-gray-200">

<div class="flex items-center">

<div class="w-10 h-10 bg-gray-400 rounded-lg flex items-center justify-center mr-3">

<svg class="w-5 h-5 text-white" fill="currentColor" viewBox="0 0 24 24">

<path d="M12.5 8.5v7h-1v-7h1zm-8 7V12a4 4 0 014-4h8a4 4 0 014 4v3.5h-1V12a3 3 0 00-3-3h-8a3 3 0 00-3 3v3.5h-1z"/>

</svg>

</div>

<div>

<h3 class="text-lg font-semibold text-gray-900">No Manure Data</h3>

<p class="text-sm text-gray-600">Manure production data is not available for this LPA area.</p>

</div>

</div>

</div>

</div>

`;

}

}

// Quick Actions Section (always shown) - exactly same as LAD

html += `

<div class="bg-white border rounded-lg">

<button class="w-full text-left px-4 py-3 font-semibold text-gray-800 hover:bg-gray-50" onclick="this.nextElementSibling.style.display = this.nextElementSibling.style.display === 'none' ? 'block' : 'none'">

⚡ Quick Actions

</button>

<div style="display: none;" class="border-t p-3">

<div class="space-y-2">

<button onclick="exportLPAData('${lpaCode}', '${lpaName.replace(/'/g, "\\'")}', ${totalPlants}, ${JSON.stringify(lpaManureEntry || {}).replace(/"/g, '&quot;')})"

class="w-full bg-blue-600 text-white px-3 py-2 rounded text-sm hover:bg-blue-700 transition-colors">

📄 Export LPA Report

</button>

<button onclick="showLPAOnMap('${lpaCode}')"

class="w-full bg-green-600 text-white px-3 py-2 rounded text-sm hover:bg-green-700 transition-colors">

🗺️ Center on Map

</button>

<button onclick="compareLPAToRegion('${lpaCode}', '${lpaName.replace(/'/g, "\\'")}', ${totalPlants}, ${totalCapacity})"

class="w-full bg-purple-600 text-white px-3 py-2 rounded text-sm hover:bg-purple-700 transition-colors">

📈 Regional Comparison

</button>

</div>

</div>

</div>

`;

html += `</div>`;

content.innerHTML = html;

// Show the panel

panel.classList.remove('translate-x-full');

panel.classList.add('translate-x-0');

console.log('LPA info panel displayed successfully');

}

// Helper function to group plants by status

function getPlantsByStatus(plants) {

const statusCounts = {};

plants.forEach(plant => {

const status = plant.Status || 'Unknown';

statusCounts[status] = (statusCounts[status] || 0) + 1;

});

return Object.entries(statusCounts).map(([status, count]) => ({

status,

count

})).sort((a, b) => b.count - a.count);

}

// ==================== QUICK ACTIONS FUNCTIONS ====================

function exportLADData(ladCode, ladName, totalPlants, manureData) {

const data = {

lad\_code: ladCode,

lad\_name: ladName,

total\_plants: totalPlants,

manure\_data: manureData,

export\_date: new Date().toISOString()

};

const csvContent = `LAD Code,LAD Name,Total Plants,Export Date\n${ladCode},"${ladName}",${totalPlants},"${data.export\_date}"`;

const blob = new Blob([csvContent], { type: 'text/csv' });

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

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

a.href = url;

a.download = `LAD\_${ladCode}\_${ladName.replace(/[^a-zA-Z0-9]/g, '\_')}\_report.csv`;

a.click();

window.URL.revokeObjectURL(url);

}

function exportLPAData(lpaCode, lpaName, totalPlants, manureData) {

const data = {

lpa\_code: lpaCode,

lpa\_name: lpaName,

total\_plants: totalPlants,

manure\_data: manureData,

export\_date: new Date().toISOString()

};

const csvContent = `LPA Code,LPA Name,Total Plants,Export Date\n${lpaCode},"${lpaName}",${totalPlants},"${data.export\_date}"`;

const blob = new Blob([csvContent], { type: 'text/csv' });

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

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

a.href = url;

a.download = `LPA\_${lpaCode}\_${lpaName.replace(/[^a-zA-Z0-9]/g, '\_')}\_report.csv`;

a.click();

window.URL.revokeObjectURL(url);

}

function showLADOnMap(ladCode) {

console.log(`Centering map on LAD: ${ladCode}`);

// Find the LAD feature and center map on it

if (window.ladBoundaries && window.ladBoundaries.features) {

const ladFeature = window.ladBoundaries.features.find(f => f.properties.LAD23CD === ladCode);

if (ladFeature && ladFeature.properties.LAT && ladFeature.properties.LONG) {

map.flyTo({

center: [ladFeature.properties.LONG, ladFeature.properties.LAT],

zoom: 10,

duration: 2000

});

}

}

}

function showLPAOnMap(lpaCode) {

console.log(`Centering map on LPA: ${lpaCode}`);

// Find the LPA feature and center map on it

if (window.lpaBoundaries && window.lpaBoundaries.features) {

const lpaFeature = window.lpaBoundaries.features.find(f => f.properties.LPA23CD === lpaCode);

if (lpaFeature && lpaFeature.properties.LAT && lpaFeature.properties.LONG) {

map.flyTo({

center: [lpaFeature.properties.LONG, lpaFeature.properties.LAT],

zoom: 10,

duration: 2000

});

}

}

}

function flyToPlant(latitude, longitude, plantName) {

console.log(`Flying to plant: ${plantName} at ${latitude}, ${longitude}`);

if (latitude && longitude && map) {

// Close the info panel first for better view

closeInfoPanel();

// Fly to the plant location

map.flyTo({

center: [longitude, latitude],

zoom: 15, // Closer zoom to see the plant

duration: 2000

});

// Optional: Show a popup at the plant location

setTimeout(() => {

if (map.getLayer('popup-marker')) {

map.removeLayer('popup-marker');

map.removeSource('popup-marker');

}

map.addSource('popup-marker', {

type: 'geojson',

data: {

type: 'Feature',

geometry: {

type: 'Point',

coordinates: [longitude, latitude]

},

properties: {

name: plantName

}

}

});

map.addLayer({

id: 'popup-marker',

type: 'circle',

source: 'popup-marker',

paint: {

'circle-radius': 8,

'circle-color': '#ff4444',

'circle-stroke-width': 2,

'circle-stroke-color': '#ffffff'

}

});

// Remove the marker after 3 seconds

setTimeout(() => {

if (map.getLayer('popup-marker')) {

map.removeLayer('popup-marker');

map.removeSource('popup-marker');

}

}, 3000);

}, 2100);

}

}

function compareLADToRegion(ladCode, ladName, totalPlants, totalCapacity) {

console.log(`Regional comparison for LAD: ${ladName}`);

// Simple regional statistics - could be enhanced with more data

const averagePlantsPerLAD = 2.5; // England/Wales average

const averageCapacityPerLAD = 1200; // kWe average

const comparison = {

plants\_vs\_average: totalPlants / averagePlantsPerLAD,

capacity\_vs\_average: totalCapacity / averageCapacityPerLAD

};

const modal = document.createElement('div');

modal.className = 'fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50';

modal.innerHTML = `

<div class="bg-white rounded-lg p-6 max-w-md mx-4">

<h3 class="text-lg font-bold mb-4">Regional Comparison: ${ladName}</h3>

<div class="space-y-3">

<div>

<div class="text-sm text-gray-600">Plants vs Regional Average</div>

<div class="font-medium ${comparison.plants\_vs\_average > 1 ? 'text-green-600' : 'text-red-600'}">

${Math.round(comparison.plants\_vs\_average \* 100)}% of average

</div>

</div>

<div>

<div class="text-sm text-gray-600">Capacity vs Regional Average</div>

<div class="font-medium ${comparison.capacity\_vs\_average > 1 ? 'text-green-600' : 'text-red-600'}">

${Math.round(comparison.capacity\_vs\_average \* 100)}% of average

</div>

</div>

</div>

<button onclick="this.closest('.fixed').remove()"

class="mt-4 w-full bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Close

</button>

</div>

`;

document.body.appendChild(modal);

}

function compareLPAToRegion(lpaCode, lpaName, totalPlants, totalCapacity) {

console.log(`Regional comparison for LPA: ${lpaName}`);

// Simple regional statistics - could be enhanced with more data

const averagePlantsPerLPA = 2.3; // England/Wales average for LPAs

const averageCapacityPerLPA = 1100; // kWe average for LPAs

const comparison = {

plants\_vs\_average: totalPlants / averagePlantsPerLPA,

capacity\_vs\_average: totalCapacity / averageCapacityPerLPA

};

const modal = document.createElement('div');

modal.className = 'fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50';

modal.innerHTML = `

<div class="bg-white rounded-lg p-6 max-w-md mx-4">

<h3 class="text-lg font-bold mb-4">Regional Comparison: ${lpaName}</h3>

<div class="space-y-3">

<div>

<div class="text-sm text-gray-600">Plants vs Regional Average</div>

<div class="font-medium ${comparison.plants\_vs\_average > 1 ? 'text-green-600' : 'text-red-600'}">

${Math.round(comparison.plants\_vs\_average \* 100)}% of average

</div>

</div>

<div>

<div class="text-sm text-gray-600">Capacity vs Regional Average</div>

<div class="font-medium ${comparison.capacity\_vs\_average > 1 ? 'text-green-600' : 'text-red-600'}">

${Math.round(comparison.capacity\_vs\_average \* 100)}% of average

</div>

</div>

</div>

<button onclick="this.closest('.fixed').remove()"

class="mt-4 w-full bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Close

</button>

</div>

`;

document.body.appendChild(modal);

}

// ==================== TOP 10 MODAL FUNCTIONS ====================

async function openTop10Modal() {

const modal = document.getElementById('top10-modal');

if (!modal) {

console.error('Top 10 modal not found in DOM');

return;

}

// Show modal immediately with loading state

modal.classList.remove('hidden');

// Show loading state

const content = document.getElementById('top10-content');

if (content) {

content.innerHTML = `

<div class="text-center p-8">

<div class="animate-spin rounded-full h-8 w-8 border-b-2 border-blue-600 mx-auto mb-4"></div>

<div class="text-slate-600">Loading data for Top 10 rankings...</div>

</div>

`;

}

try {

// Ensure AD plants data is loaded

if (!adPlantsData || adPlantsData.length === 0) {

console.log('Loading AD plants data for Top 10 rankings...');

await loadADPlants();

}

// Verify data loaded successfully

if (!adPlantsData || adPlantsData.length === 0) {

throw new Error('Failed to load AD plants data');

}

console.log(`Top 10 modal opened with ${adPlantsData.length} plants loaded`);

showTop10Tab('plants-gwh');

} catch (error) {

console.error('Error opening Top 10 modal:', error);

if (content) {

content.innerHTML = `

<div class="text-center p-8">

<div class="text-red-600 mb-4">Error loading data</div>

<div class="text-sm text-slate-500 mb-4">${error.message}</div>

<button onclick="openTop10Modal()" class="bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Try Again

</button>

</div>

`;

}

}

}

function closeTop10Modal() {

const modal = document.getElementById('top10-modal');

modal.classList.add('hidden');

}

function showTop10Tab(tabName) {

// Update tab buttons - new styling system

document.querySelectorAll('.top10-tab').forEach(tab => {

tab.className = 'top10-tab px-4 py-2 rounded-lg border-2 font-medium text-sm transition-all duration-200 border-slate-300 bg-white text-slate-600 hover:border-blue-400 hover:text-blue-600 hover:shadow-sm';

});

const targetTab = document.querySelector(`[data-tab="${tabName}"]`);

if (targetTab) {

targetTab.className = 'top10-tab px-4 py-2 rounded-lg border-2 font-medium text-sm transition-all duration-200 border-blue-500 bg-blue-500 text-white shadow-md';

}

// Update content

const content = document.getElementById('top10-content');

if (!content) {

console.error('Top 10 content container not found');

return;

}

try {

switch (tabName) {

case 'plants-gwh':

content.innerHTML = renderTop10PlantsByGWh();

break;

case 'plants-capacity':

content.innerHTML = renderTop10PlantsByCapacity();

break;

case 'lad-manure':

content.innerHTML = renderTop10LADsByManure();

break;

case 'lad-manure-types':

content.innerHTML = renderTop10LADsByManureTypes();

break;

case 'lads-by-plant-count':

content.innerHTML = renderTop10LADsByPlantCount();

break;

case 'lpas-by-plant-count':

content.innerHTML = renderTop10LPAsByPlantCount();

break;

case 'plants-operational-gtg':

content.innerHTML = renderTop10OperationalPlantsByGWh();

break;

case 'plants-operational-chp':

content.innerHTML = renderTop10OperationalPlantsByCapacity();

break;

case 'lads-by-gwh':

content.innerHTML = renderTop10LADsByGWh();

break;

case 'lads-by-kwe':

content.innerHTML = renderTop10LADsByKWe();

break;

default:

content.innerHTML = `

<div class="text-center p-8">

<div class="text-slate-600">Unknown tab: ${tabName}</div>

<button onclick="showTop10Tab('plants-gwh')" class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Go to Plants by GWh

</button>

</div>

`;

}

console.log(`Top 10 tab '${tabName}' rendered successfully`);

} catch (error) {

console.error(`Error rendering Top 10 tab '${tabName}':`, error);

content.innerHTML = `

<div class="text-center p-8">

<div class="text-red-600">Error loading ${tabName}</div>

<div class="text-sm text-slate-500 mt-2">${error.message}</div>

<button onclick="showTop10Tab('plants-gwh')" class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Try Default Tab

</button>

</div>

`;

}

}

function renderTop10PlantsByGWh() {

if (!adPlantsData || adPlantsData.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">Loading AD plants data...</div>

<button onclick="loadADPlants().then(() => showTop10Tab('plants-gwh'))"

class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Reload Plants Data

</button>

</div>

`;

}

const plantsByGWh = adPlantsData

.filter(plant => {

const gwh = parseFloat(plant.GWh) || 0;

return gwh > 0;

})

.sort((a, b) => (parseFloat(b.GWh) || 0) - (parseFloat(a.GWh) || 0))

.slice(0, 10);

if (plantsByGWh.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">No plants with GWh data found in the dataset.</div>

<div class="text-xs text-slate-500 mt-2">Total plants loaded: ${adPlantsData.length}</div>

</div>

`;

}

return `

<div class="grid grid-cols-1 lg:grid-cols-2 gap-6">

<div>

<h3 class="text-lg font-semibold text-slate-900 mb-4">

Top 10 Plants by Annual Generation

</h3>

<div class="bg-blue-50 p-3 rounded-lg mb-4 text-sm">

<div class="flex justify-between">

<span class="text-slate-600">Total plants with GWh data:</span>

<span class="font-semibold">${adPlantsData.filter(p => parseFloat(p.GWh) > 0).length}</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">Total generation (Top 10):</span>

<span class="font-semibold text-green-600">${formatNumber(plantsByGWh.reduce((sum, p) => sum + (parseFloat(p.GWh) || 0), 0))} GWh</span>

</div>

</div>

<div class="space-y-3">

${plantsByGWh.map((plant, index) => `

<div class="bg-white rounded-lg border border-slate-200 p-4 cursor-pointer hover:bg-slate-50 transition-colors plant-card"

onclick="flyToPlant(${plant.Latitude}, ${plant.Longitude}, '${(plant['Site name'] || plant['Site Name'] || 'Unknown Site').replace(/'/g, "\\'")}')">

<div class="flex items-center justify-between">

<div class="flex items-center">

<div class="w-8 h-8 bg-green-600 text-white rounded-full flex items-center justify-center text-sm font-semibold mr-3">

${index + 1}

</div>

<div>

<div class="font-medium text-slate-900">${plant['Site name'] || plant['Site Name'] || 'Unknown Site'}</div>

<div class="text-sm text-slate-600">${plant.Developer || 'Unknown'} • ${plant.Status || 'Unknown'}</div>

<div class="text-xs text-slate-500">${plant.lad\_name || plant['LAD name'] || plant.County || 'Unknown Location'}</div>

</div>

</div>

<div class="text-right">

<div class="font-bold text-green-600">

${formatNumber(parseFloat(plant.GWh) || 0)} GWh/yr

</div>

<div class="text-sm text-slate-600">

${formatNumber(parseFloat(String(plant['Capacity(kWe)']).replace(/,/g, '')) || 0)} kWe

</div>

<div class="text-xs text-slate-500">

${plant.Type || 'Unknown Type'}

</div>

</div>

</div>

</div>

`).join('')}

</div>

</div>

<div>

<h3 class="text-lg font-semibold text-slate-900 mb-4">

Generation Analysis

</h3>

<div class="bg-green-50 p-4 rounded-lg">

<div class="space-y-3 text-sm">

<div class="flex justify-between">

<span class="text-slate-600">Highest Generator:</span>

<span class="font-semibold">${plantsByGWh[0]['Site name'] || plantsByGWh[0]['Site Name'] || 'Unknown'}</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">Peak Generation:</span>

<span class="font-semibold text-green-600">${formatNumber(parseFloat(plantsByGWh[0].GWh) || 0)} GWh/yr</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">Average (Top 10):</span>

<span class="font-semibold">${formatNumber(plantsByGWh.reduce((sum, p) => sum + (parseFloat(p.GWh) || 0), 0) / plantsByGWh.length)} GWh/yr</span>

</div>

<div class="pt-2 border-t border-green-200 text-xs text-slate-500">

Click on any plant to view its location on the map

</div>

</div>

</div>

</div>

</div>

`;

}

function renderTop10PlantsByCapacity() {

if (!adPlantsData || adPlantsData.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">Loading AD plants data...</div>

<button onclick="loadADPlants().then(() => showTop10Tab('plants-capacity'))"

class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Reload Plants Data

</button>

</div>

`;

}

const plantsByCapacity = adPlantsData

.filter(plant => {

const capacity = parseFloat(String(plant['Capacity(kWe)']).replace(/,/g, '')) || 0;

return capacity > 0;

})

.sort((a, b) => {

const capacityA = parseFloat(String(a['Capacity(kWe)']).replace(/,/g, '')) || 0;

const capacityB = parseFloat(String(b['Capacity(kWe)']).replace(/,/g, '')) || 0;

return capacityB - capacityA;

})

.slice(0, 10);

if (plantsByCapacity.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">No plants with capacity data found in the dataset.</div>

<div class="text-xs text-slate-500 mt-2">Total plants loaded: ${adPlantsData.length}</div>

</div>

`;

}

const totalCapacity = plantsByCapacity.reduce((sum, plant) => {

return sum + (parseFloat(String(plant['Capacity(kWe)']).replace(/,/g, '')) || 0);

}, 0);

return `

<div class="grid grid-cols-1 lg:grid-cols-2 gap-6">

<div>

<h3 class="text-lg font-semibold text-slate-900 mb-4">

Top 10 Plants by Electrical Capacity

</h3>

<div class="bg-blue-50 p-3 rounded-lg mb-4 text-sm">

<div class="flex justify-between">

<span class="text-slate-600">Total plants with capacity data:</span>

<span class="font-semibold">${adPlantsData.filter(p => parseFloat(String(p['Capacity(kWe)']).replace(/,/g, '')) > 0).length}</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">Total capacity (Top 10):</span>

<span class="font-semibold text-blue-600">${formatNumber(totalCapacity)} kWe</span>

</div>

</div>

<div class="space-y-3">

${plantsByCapacity.map((plant, index) => {

const capacity = parseFloat(String(plant['Capacity(kWe)']).replace(/,/g, '')) || 0;

const gwh = parseFloat(plant.GWh) || 0;

return `

<div class="bg-white rounded-lg border border-slate-200 p-4 cursor-pointer hover:bg-slate-50 transition-colors plant-card"

onclick="flyToPlant(${plant.Latitude}, ${plant.Longitude}, '${(plant['Site name'] || plant['Site Name'] || 'Unknown Site').replace(/'/g, "\\'")}')">

<div class="flex items-center justify-between">

<div class="flex items-center">

<div class="w-8 h-8 bg-blue-600 text-white rounded-full flex items-center justify-center text-sm font-semibold mr-3">

${index + 1}

</div>

<div>

<div class="font-medium text-slate-900">${plant['Site name'] || plant['Site Name'] || 'Unknown Site'}</div>

<div class="text-sm text-slate-600">${plant.Developer || 'Unknown'} • ${plant.Status || 'Unknown'}</div>

<div class="text-xs text-slate-500">${plant.lad\_name || plant['LAD name'] || plant.County || 'Unknown Location'}</div>

</div>

</div>

<div class="text-right">

<div class="font-bold text-blue-600">

${formatNumber(capacity)} kWe

</div>

<div class="text-sm text-green-600">

${gwh > 0 ? formatNumber(gwh) + ' GWh/yr' : 'GWh not specified'}

</div>

<div class="text-xs text-slate-500">

${plant.Type || 'Unknown Type'}

</div>

</div>

</div>

</div>

`;

}).join('')}

</div>

</div>

<div>

<h3 class="text-lg font-semibold text-slate-900 mb-4">

Capacity Analysis

</h3>

<div class="bg-blue-50 p-4 rounded-lg">

<div class="space-y-3 text-sm">

<div class="flex justify-between">

<span class="text-slate-600">Largest Plant:</span>

<span class="font-semibold">${plantsByCapacity[0]['Site name'] || plantsByCapacity[0]['Site Name'] || 'Unknown'}</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">Peak Capacity:</span>

<span class="font-semibold text-blue-600">${formatNumber(parseFloat(String(plantsByCapacity[0]['Capacity(kWe)']).replace(/,/g, '')) || 0)} kWe</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">Average (Top 10):</span>

<span class="font-semibold">${formatNumber(totalCapacity / plantsByCapacity.length)} kWe</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">Total (MWe):</span>

<span class="font-semibold text-indigo-600">${formatNumber(totalCapacity / 1000)} MWe</span>

</div>

<div class="pt-2 border-t border-blue-200 text-xs text-slate-500">

Click on any plant to navigate to its location on the map

</div>

</div>

</div>

</div>

</div>

`;

}

function renderTop10LADsByManure() {

if (!window.ladManureData || !window.ladManureData.length) {

return `

<div class="text-center p-8">

<div class="text-slate-600">Loading manure data...</div>

</div>

`;

}

const ladsByManure = window.ladManureData

.map(lad => {

// Calculate total tonnes from all manure types

const totalTonnes = parseFloat(String(lad.total\_tonnes).replace(/,/g, '').trim()) || 0;

const totalGWh = parseFloat(String(lad.total\_gwh).replace(/,/g, '').trim()) || 0;

return {

...lad,

totalManure: totalTonnes,

totalGWhPotential: totalGWh

};

})

.filter(lad => lad.totalManure > 0)

.sort((a, b) => b.totalManure - a.totalManure)

.slice(0, 10);

return `

<div class="grid grid-cols-1 lg:grid-cols-2 gap-6">

<div>

<h3 class="text-lg font-semibold text-slate-900 mb-4">

Top 10 LADs by Total Manure Production

</h3>

<div class="space-y-3">

${ladsByManure.map((lad, index) => `

<div class="bg-white rounded-lg border border-slate-200 p-4 cursor-pointer hover:bg-amber-50 transition-colors"

onclick="showLADOnMap('${lad.lad\_code}')">

<div class="flex items-center justify-between">

<div class="flex items-center">

<div class="w-8 h-8 bg-amber-600 text-white rounded-full flex items-center justify-center text-sm font-semibold mr-3">

${index + 1}

</div>

<div>

<div class="font-medium text-slate-900">${lad.lad\_name}</div>

<div class="text-sm text-slate-600">Code: ${lad.lad\_code}</div>

</div>

</div>

<div class="text-right">

<div class="font-bold text-amber-600">

${formatNumber(Math.round(lad.totalManure))} tonnes/yr

</div>

<div class="text-sm text-green-600">

${formatNumber(Math.round(lad.totalGWhPotential))} GWh potential

</div>

</div>

</div>

</div>

`).join('')}

</div>

</div>

<div>

<h3 class="text-lg font-semibold text-slate-900 mb-4">

Manure Production Summary

</h3>

<div class="bg-amber-50 p-4 rounded-lg">

<div class="space-y-3 text-sm">

<div class="flex justify-between">

<span class="text-slate-600">Total Top 10 Production:</span>

<span class="font-semibold">${formatNumber(ladsByManure.reduce((sum, lad) => sum + lad.totalManure, 0))} tonnes/yr</span>

</div>

<div class="flex justify-between">

<span class="text-slate-600">Total Energy Potential:</span>

<span class="font-semibold text-green-600">${formatNumber(ladsByManure.reduce((sum, lad) => sum + lad.totalGWhPotential, 0))} GWh/yr</span>

</div>

<div class="pt-2 border-t border-amber-200 text-xs text-slate-500">

Click on any LAD to view it on the map

</div>

</div>

</div>

</div>

</div>

`;

}

function renderTop10LADsByManureTypes() {

if (!window.ladManureData || !window.ladManureData.length) {

return `

<div class="text-center p-8">

<div class="text-slate-600">Loading manure data...</div>

</div>

`;

}

// Define manure type configurations with correct field names

const manureTypes = [

{

field: 'Beef FYM\_tonnes',

gwhField: 'Beef FYM\_gwh',

name: 'Beef FYM',

color: '#8B4513'

},

{

field: 'Beef Slurry\_tonnes',

gwhField: 'Beef Slurry\_gwh',

name: 'Beef Slurry',

color: '#A0522D'

},

{

field: 'Dairy FYM\_tonnes',

gwhField: 'Dairy FYM\_gwh',

name: 'Dairy FYM',

color: '#4A90E2'

},

{

field: 'Dairy Slurry\_tonnes',

gwhField: 'Dairy Slurry\_gwh',

name: 'Dairy Slurry',

color: '#7BB3F0'

},

{

field: 'Broilers\_tonnes',

gwhField: 'Broilers\_gwh',

name: 'Broilers',

color: '#FFB347'

},

{

field: 'Layers\_tonnes',

gwhField: 'Layers\_gwh',

name: 'Layers',

color: '#FFD700'

},

{

field: 'Pigs\_tonnes',

gwhField: 'Pigs\_gwh',

name: 'Pigs',

color: '#FF69B4'

},

{

field: 'Sheep\_tonnes',

gwhField: 'Sheep\_gwh',

name: 'Sheep',

color: '#98FB98'

}

];

// Calculate totals for each manure type across all LADs for percentage calculations

const manureTotals = {};

manureTypes.forEach(type => {

manureTotals[type.field] = window.ladManureData.reduce((sum, lad) => {

return sum + (parseFloat(String(lad[type.field]).replace(/,/g, '').trim()) || 0);

}, 0);

});

return `

<div class="grid grid-cols-1 lg:grid-cols-2 gap-6">

<div>

<h3 class="text-lg font-semibold text-slate-900 mb-4">

Top 10 LADs by Manure Type (Part 1)

</h3>

<div class="space-y-4">

${manureTypes.slice(0, 4).map(config => {

const topLADs = window.ladManureData

.map(lad => ({

...lad,

value: parseFloat(String(lad[config.field]).replace(/,/g, '').trim()) || 0,

gwhValue: parseFloat(String(lad[config.gwhField]).replace(/,/g, '').trim()) || 0

}))

.filter(lad => lad.value > 0)

.sort((a, b) => b.value - a.value)

.slice(0, 10);

const totalForType = manureTotals[config.field];

return `

<div class="bg-white border border-slate-200 rounded-lg p-3">

<h4 class="font-medium text-slate-800 text-sm flex items-center justify-between mb-2">

<div class="flex items-center">

<div class="w-3 h-3 rounded mr-2" style="background-color: ${config.color}"></div>

${config.name}

</div>

<div class="text-xs text-slate-500">

Total: ${formatNumber(Math.round(totalForType))} t/yr

</div>

</h4>

<div class="space-y-1 max-h-64 overflow-y-auto">

${topLADs.map((lad, index) => {

const percentage = ((lad.value / totalForType) \* 100);

return `

<div class="flex items-center justify-between p-2 bg-slate-50 rounded text-xs hover:bg-slate-100 cursor-pointer transition-colors"

onclick="showLADOnMap('${lad.lad\_code}')">

<div class="flex items-center min-w-0 flex-1">

<span class="w-5 h-5 bg-slate-600 text-white rounded-full flex items-center justify-center text-xs font-semibold mr-2 flex-shrink-0">

${index + 1}

</span>

<span class="font-medium truncate">${lad.lad\_name}</span>

</div>

<div class="text-right ml-2 flex-shrink-0">

<div class="font-semibold" style="color: ${config.color}">

${formatNumber(Math.round(lad.value))} t/yr

</div>

<div class="text-green-600 font-medium">

${formatNumber(Math.round(lad.gwhValue))} GWh

</div>

<div class="text-slate-500">

${percentage.toFixed(1)}% of total

</div>

</div>

</div>

`;

}).join('')}

</div>

</div>

`;

}).join('')}

</div>

</div>

<div>

<h3 class="text-lg font-semibold text-slate-900 mb-4">

Top 10 LADs by Manure Type (Part 2)

</h3>

<div class="space-y-4">

${manureTypes.slice(4).map(config => {

const topLADs = window.ladManureData

.map(lad => ({

...lad,

value: parseFloat(String(lad[config.field]).replace(/,/g, '').trim()) || 0,

gwhValue: parseFloat(String(lad[config.gwhField]).replace(/,/g, '').trim()) || 0

}))

.filter(lad => lad.value > 0)

.sort((a, b) => b.value - a.value)

.slice(0, 10);

const totalForType = manureTotals[config.field];

return `

<div class="bg-white border border-slate-200 rounded-lg p-3">

<h4 class="font-medium text-slate-800 text-sm flex items-center justify-between mb-2">

<div class="flex items-center">

<div class="w-3 h-3 rounded mr-2" style="background-color: ${config.color}"></div>

${config.name}

</div>

<div class="text-xs text-slate-500">

Total: ${formatNumber(Math.round(totalForType))} t/yr

</div>

</h4>

<div class="space-y-1 max-h-64 overflow-y-auto">

${topLADs.map((lad, index) => {

const percentage = ((lad.value / totalForType) \* 100);

return `

<div class="flex items-center justify-between p-2 bg-slate-50 rounded text-xs hover:bg-slate-100 cursor-pointer transition-colors"

onclick="showLADOnMap('${lad.lad\_code}')">

<div class="flex items-center min-w-0 flex-1">

<span class="w-5 h-5 bg-slate-600 text-white rounded-full flex items-center justify-center text-xs font-semibold mr-2 flex-shrink-0">

${index + 1}

</span>

<span class="font-medium truncate">${lad.lad\_name}</span>

</div>

<div class="text-right ml-2 flex-shrink-0">

<div class="font-semibold" style="color: ${config.color}">

${formatNumber(Math.round(lad.value))} t/yr

</div>

<div class="text-green-600 font-medium">

${formatNumber(Math.round(lad.gwhValue))} GWh

</div>

<div class="text-slate-500">

${percentage.toFixed(1)}% of total

</div>

</div>

</div>

`;

}).join('')}

</div>

</div>

`;

}).join('')}

</div>

</div>

</div>

`;

}

function renderTop10LADsByPlantCount() {

if (!adPlantsData || adPlantsData.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">Loading AD plants data...</div>

<button onclick="loadADPlants().then(() => showTop10Tab('lads-by-plant-count'))"

class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Reload Plants Data

</button>

</div>

`;

}

// Group plants by LAD and count them

const ladCounts = {};

adPlantsData.forEach(plant => {

const ladCode = plant.lad\_code;

const ladName = plant.lad\_name;

if (ladCode && ladName) {

if (!ladCounts[ladCode]) {

ladCounts[ladCode] = {

lad\_code: ladCode,

lad\_name: ladName,

count: 0,

total\_capacity: 0,

operational\_count: 0

};

}

ladCounts[ladCode].count++;

const capacity = parseFloat(plant['Capacity(kWe)']) || 0;

ladCounts[ladCode].total\_capacity += capacity;

if (plant.Group && (plant.Group.includes('Operational') || plant.Group === 'Under Construction')) {

ladCounts[ladCode].operational\_count++;

}

}

});

const topLADs = Object.values(ladCounts)

.filter(lad => lad.count > 0)

.sort((a, b) => b.count - a.count)

.slice(0, 10);

if (topLADs.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">No LADs with plants found in the dataset.</div>

</div>

`;

}

const totalPlants = Object.values(ladCounts).reduce((sum, lad) => sum + lad.count, 0);

return `

<div class="space-y-4">

<div class="bg-gradient-to-r from-blue-50 to-indigo-50 rounded-lg p-4">

<h3 class="text-lg font-semibold text-slate-900 mb-2">

Top 10 LADs by Number of AD Plants

</h3>

<div class="grid grid-cols-2 md:grid-cols-4 gap-4 text-sm">

<div class="text-center">

<div class="text-2xl font-bold text-blue-600">${totalPlants}</div>

<div class="text-slate-600">Total Plants</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-green-600">${topLADs.reduce((sum, lad) => sum + lad.count, 0)}</div>

<div class="text-slate-600">Top 10 Total</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-amber-600">${Math.round(topLADs.reduce((sum, lad) => sum + lad.count, 0) / topLADs.length)}</div>

<div class="text-slate-600">Average (Top 10)</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-purple-600">${Math.round((topLADs.reduce((sum, lad) => sum + lad.count, 0) / totalPlants) \* 100)}%</div>

<div class="text-slate-600">% of Total</div>

</div>

</div>

</div>

<div class="grid gap-3">

${topLADs.map((lad, index) => {

const percentage = ((lad.count / totalPlants) \* 100);

const operationalPercentage = ((lad.operational\_count / lad.count) \* 100);

return `

<div class="bg-white border border-slate-200 rounded-lg p-4 hover:shadow-md transition-shadow cursor-pointer"

onclick="showLADOnMap('${lad.lad\_code}')">

<div class="flex items-center justify-between">

<div class="flex items-center min-w-0 flex-1">

<span class="w-8 h-8 bg-gradient-to-r from-blue-500 to-indigo-600 text-white rounded-full flex items-center justify-center text-sm font-bold mr-3 flex-shrink-0">

${index + 1}

</span>

<div class="min-w-0 flex-1">

<h4 class="font-semibold text-slate-900 truncate">${lad.lad\_name}</h4>

<p class="text-sm text-slate-600">LAD Code: ${lad.lad\_code}</p>

</div>

</div>

<div class="text-right ml-4 flex-shrink-0">

<div class="text-2xl font-bold text-blue-600">${lad.count}</div>

<div class="text-sm text-slate-600">Plants</div>

</div>

</div>

<div class="mt-3 grid grid-cols-3 gap-3 text-sm">

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-green-600">${lad.operational\_count}</div>

<div class="text-xs text-slate-600">Operational</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-amber-600">${formatNumber(Math.round(lad.total\_capacity))} kWe</div>

<div class="text-xs text-slate-600">Total Capacity</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-purple-600">${percentage.toFixed(1)}%</div>

<div class="text-xs text-slate-600">of UK Total</div>

</div>

</div>

<div class="mt-2">

<div class="flex justify-between text-xs text-slate-600 mb-1">

<span>Operational Rate</span>

<span>${operationalPercentage.toFixed(1)}%</span>

</div>

<div class="w-full bg-slate-200 rounded-full h-2">

<div class="bg-gradient-to-r from-green-400 to-green-600 h-2 rounded-full transition-all duration-300"

style="width: ${operationalPercentage}%"></div>

</div>

</div>

</div>

`;

}).join('')}

</div>

</div>

`;

}

function renderTop10PlantsByGroup(groupName) {

if (!adPlantsData || adPlantsData.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">Loading AD plants data...</div>

<button onclick="loadADPlants().then(() => showTop10Tab('plants-operational-${groupName.includes('GtG') ? 'gtg' : 'chp'}'))"

class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Reload Plants Data

</button>

</div>

`;

}

const plantsInGroup = adPlantsData

.filter(plant => plant.Group === groupName)

.sort((a, b) => (parseFloat(b['Capacity(kWe)']) || 0) - (parseFloat(a['Capacity(kWe)']) || 0))

.slice(0, 10);

if (plantsInGroup.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">No plants found in group: ${groupName}</div>

<div class="text-xs text-slate-500 mt-2">Total plants loaded: ${adPlantsData.length}</div>

</div>

`;

}

const totalGroupCapacity = adPlantsData

.filter(plant => plant.Group === groupName)

.reduce((sum, plant) => sum + (parseFloat(plant['Capacity(kWe)']) || 0), 0);

const groupColor = groupName.includes('GtG') ? '#0f5132' : '#198754';

return `

<div class="space-y-4">

<div class="bg-gradient-to-r from-green-50 to-emerald-50 rounded-lg p-4">

<h3 class="text-lg font-semibold text-slate-900 mb-2">

Top 10 Plants: ${groupName}

</h3>

<div class="grid grid-cols-2 md:grid-cols-4 gap-4 text-sm">

<div class="text-center">

<div class="text-2xl font-bold" style="color: ${groupColor}">${adPlantsData.filter(p => p.Group === groupName).length}</div>

<div class="text-slate-600">Total in Group</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-blue-600">${formatNumber(Math.round(totalGroupCapacity))}</div>

<div class="text-slate-600">Total Capacity (kWe)</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-amber-600">${formatNumber(Math.round(plantsInGroup.reduce((sum, plant) => sum + (parseFloat(plant['Capacity(kWe)']) || 0), 0) / plantsInGroup.length))}</div>

<div class="text-slate-600">Average (Top 10)</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-purple-600">${Math.round((plantsInGroup.reduce((sum, plant) => sum + (parseFloat(plant['Capacity(kWe)']) || 0), 0) / totalGroupCapacity) \* 100)}%</div>

<div class="text-slate-600">% of Group Total</div>

</div>

</div>

</div>

<div class="grid gap-3">

${plantsInGroup.map((plant, index) => {

const capacity = parseFloat(plant['Capacity(kWe)']) || 0;

const percentage = ((capacity / totalGroupCapacity) \* 100);

return `

<div class="bg-white border border-slate-200 rounded-lg p-4 hover:shadow-md transition-shadow cursor-pointer"

onclick="selectPlantFromTop10(${plant.Latitude}, ${plant.Longitude}, ${JSON.stringify(plant).replace(/"/g, '&quot;')})">

<div class="flex items-center justify-between">

<div class="flex items-center min-w-0 flex-1">

<span class="w-8 h-8 rounded-full flex items-center justify-center text-sm font-bold mr-3 flex-shrink-0 text-white"

style="background: linear-gradient(135deg, ${groupColor}, ${groupColor}99)">

${index + 1}

</span>

<div class="min-w-0 flex-1">

<h4 class="font-semibold text-slate-900 truncate">${plant['Site name'] || 'Unknown Site'}</h4>

<p class="text-sm text-slate-600">${plant.Developer || 'Unknown Developer'}</p>

<p class="text-xs text-slate-500">${plant['lad\_name'] || plant.County || 'Unknown Location'}</p>

</div>

</div>

<div class="text-right ml-4 flex-shrink-0">

<div class="text-2xl font-bold" style="color: ${groupColor}">${formatNumber(Math.round(capacity))}</div>

<div class="text-sm text-slate-600">kWe</div>

</div>

</div>

<div class="mt-3 grid grid-cols-3 gap-3 text-sm">

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-blue-600">${plant.Status || 'Unknown'}</div>

<div class="text-xs text-slate-600">Status</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-green-600">${plant.Completion || 'N/A'}</div>

<div class="text-xs text-slate-600">Year</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-purple-600">${percentage.toFixed(1)}%</div>

<div class="text-xs text-slate-600">of Group</div>

</div>

</div>

${plant.Feedstock ? `

<div class="mt-2 text-xs text-slate-600">

<strong>Feedstock:</strong> ${plant.Feedstock.length > 60 ? plant.Feedstock.substring(0, 60) + '...' : plant.Feedstock}

</div>

` : ''}

</div>

`;

}).join('')}

</div>

</div>

`;

}

function renderTop10LADsByGWh() {

if (!adPlantsData || adPlantsData.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">Loading AD plants data...</div>

<button onclick="loadADPlants().then(() => showTop10Tab('lads-by-gwh'))"

class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Reload Plants Data

</button>

</div>

`;

}

// Group plants by LAD and calculate total GWh from Operational: GtG + GtG&CHP plants

const ladGWh = {};

adPlantsData.forEach(plant => {

const ladCode = plant.lad\_code;

const ladName = plant.lad\_name;

if (ladCode && ladName) {

if (!ladGWh[ladCode]) {

ladGWh[ladCode] = {

lad\_code: ladCode,

lad\_name: ladName,

total\_gwh: 0,

gtg\_gwh: 0,

plant\_count: 0,

gtg\_count: 0

};

}

const gwh = parseFloat(plant.GWh) || 0;

ladGWh[ladCode].plant\_count++;

ladGWh[ladCode].total\_gwh += gwh;

// Focus on GtG + GtG&CHP plants as requested

if (plant.Group === 'Operational: GtG + GtG&CHP') {

ladGWh[ladCode].gtg\_gwh += gwh;

ladGWh[ladCode].gtg\_count++;

}

}

});

const topLADs = Object.values(ladGWh)

.filter(lad => lad.gtg\_gwh > 0)

.sort((a, b) => b.gtg\_gwh - a.gtg\_gwh)

.slice(0, 10);

if (topLADs.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">No LADs with GtG + GtG&CHP plants producing GWh found.</div>

<div class="text-xs text-slate-500 mt-2">Note: Focusing on Operational: GtG + GtG&CHP output as requested</div>

</div>

`;

}

const totalGtGGWh = Object.values(ladGWh).reduce((sum, lad) => sum + lad.gtg\_gwh, 0);

return `

<div class="space-y-4">

<div class="bg-gradient-to-r from-green-50 to-emerald-50 rounded-lg p-4">

<h3 class="text-lg font-semibold text-slate-900 mb-2">

Top 10 LADs by Total GWh Output (Operational: GtG + GtG&CHP)

</h3>

<div class="grid grid-cols-2 md:grid-cols-4 gap-4 text-sm">

<div class="text-center">

<div class="text-2xl font-bold text-green-600">${formatNumber(Math.round(totalGtGGWh))}</div>

<div class="text-slate-600">Total GtG GWh</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-blue-600">${formatNumber(Math.round(topLADs.reduce((sum, lad) => sum + lad.gtg\_gwh, 0)))}</div>

<div class="text-slate-600">Top 10 GWh</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-amber-600">${Math.round(topLADs.reduce((sum, lad) => sum + lad.gtg\_gwh, 0) / topLADs.length)}</div>

<div class="text-slate-600">Average (Top 10)</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-purple-600">${Math.round((topLADs.reduce((sum, lad) => sum + lad.gtg\_gwh, 0) / totalGtGGWh) \* 100)}%</div>

<div class="text-slate-600">% of Total</div>

</div>

</div>

</div>

<div class="grid gap-3">

${topLADs.map((lad, index) => {

const percentage = ((lad.gtg\_gwh / totalGtGGWh) \* 100);

const efficiency = lad.gtg\_count > 0 ? (lad.gtg\_gwh / lad.gtg\_count) : 0;

return `

<div class="bg-white border border-slate-200 rounded-lg p-4 hover:shadow-md transition-shadow cursor-pointer"

onclick="showLADOnMap('${lad.lad\_code}')">

<div class="flex items-center justify-between">

<div class="flex items-center min-w-0 flex-1">

<span class="w-8 h-8 bg-gradient-to-r from-green-500 to-emerald-600 text-white rounded-full flex items-center justify-center text-sm font-bold mr-3 flex-shrink-0">

${index + 1}

</span>

<div class="min-w-0 flex-1">

<h4 class="font-semibold text-slate-900 truncate">${lad.lad\_name}</h4>

<p class="text-sm text-slate-600">LAD Code: ${lad.lad\_code}</p>

</div>

</div>

<div class="text-right ml-4 flex-shrink-0">

<div class="text-2xl font-bold text-green-600">${formatNumber(Math.round(lad.gtg\_gwh))}</div>

<div class="text-sm text-slate-600">GWh/year</div>

</div>

</div>

<div class="mt-3 grid grid-cols-4 gap-3 text-sm">

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-green-600">${lad.gtg\_count}</div>

<div class="text-xs text-slate-600">GtG Plants</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-blue-600">${formatNumber(Math.round(efficiency))}</div>

<div class="text-xs text-slate-600">GWh/Plant</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-amber-600">${lad.plant\_count}</div>

<div class="text-xs text-slate-600">Total Plants</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-purple-600">${percentage.toFixed(1)}%</div>

<div class="text-xs text-slate-600">of UK Total</div>

</div>

</div>

<div class="mt-2">

<div class="flex justify-between text-xs text-slate-600 mb-1">

<span>GtG Share of LAD</span>

<span>${((lad.gtg\_count / lad.plant\_count) \* 100).toFixed(1)}%</span>

</div>

<div class="w-full bg-slate-200 rounded-full h-2">

<div class="bg-gradient-to-r from-green-400 to-green-600 h-2 rounded-full transition-all duration-300"

style="width: ${Math.min(100, (lad.gtg\_count / lad.plant\_count) \* 100)}%"></div>

</div>

</div>

</div>

`;

}).join('')}

</div>

</div>

`;

}

function renderTop10LADsByKWe() {

if (!adPlantsData || adPlantsData.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">Loading AD plants data...</div>

<button onclick="loadADPlants().then(() => showTop10Tab('lads-by-kwe'))"

class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Reload Plants Data

</button>

</div>

`;

}

// Group plants by LAD and calculate total kWe capacity

const ladCapacity = {};

adPlantsData.forEach(plant => {

const ladCode = plant.lad\_code;

const ladName = plant.lad\_name;

if (ladCode && ladName) {

if (!ladCapacity[ladCode]) {

ladCapacity[ladCode] = {

lad\_code: ladCode,

lad\_name: ladName,

total\_capacity: 0,

operational\_capacity: 0,

plant\_count: 0,

operational\_count: 0,

under\_construction: 0,

planning\_capacity: 0

};

}

const capacity = parseFloat(plant['Capacity(kWe)']) || 0;

ladCapacity[ladCode].plant\_count++;

ladCapacity[ladCode].total\_capacity += capacity;

if (plant.Group && plant.Group.includes('Operational')) {

ladCapacity[ladCode].operational\_capacity += capacity;

ladCapacity[ladCode].operational\_count++;

} else if (plant.Group === 'Under Construction') {

ladCapacity[ladCode].under\_construction++;

} else if (plant.Group && plant.Group.includes('Planning')) {

ladCapacity[ladCode].planning\_capacity += capacity;

}

}

});

const topLADs = Object.values(ladCapacity)

.filter(lad => lad.total\_capacity > 0)

.sort((a, b) => b.total\_capacity - a.total\_capacity)

.slice(0, 10);

if (topLADs.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">No LADs with kWe capacity found.</div>

</div>

`;

}

const totalCapacity = Object.values(ladCapacity).reduce((sum, lad) => sum + lad.total\_capacity, 0);

return `

<div class="space-y-4">

<div class="bg-gradient-to-r from-blue-50 to-indigo-50 rounded-lg p-4">

<h3 class="text-lg font-semibold text-slate-900 mb-2">

Top 10 LADs by Total kWe Capacity (All Plants)

</h3>

<div class="grid grid-cols-2 md:grid-cols-4 gap-4 text-sm">

<div class="text-center">

<div class="text-2xl font-bold text-blue-600">${formatNumber(Math.round(totalCapacity))}</div>

<div class="text-slate-600">Total UK kWe</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-green-600">${formatNumber(Math.round(topLADs.reduce((sum, lad) => sum + lad.total\_capacity, 0)))}</div>

<div class="text-slate-600">Top 10 kWe</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-amber-600">${formatNumber(Math.round(topLADs.reduce((sum, lad) => sum + lad.total\_capacity, 0) / topLADs.length))}</div>

<div class="text-slate-600">Average (Top 10)</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-purple-600">${Math.round((topLADs.reduce((sum, lad) => sum + lad.total\_capacity, 0) / totalCapacity) \* 100)}%</div>

<div class="text-slate-600">% of Total</div>

</div>

</div>

</div>

<div class="grid gap-3">

${topLADs.map((lad, index) => {

const percentage = ((lad.total\_capacity / totalCapacity) \* 100);

const operationalPercent = ((lad.operational\_capacity / lad.total\_capacity) \* 100);

const avgPlantSize = lad.plant\_count > 0 ? (lad.total\_capacity / lad.plant\_count) : 0;

return `

<div class="bg-white border border-slate-200 rounded-lg p-4 hover:shadow-md transition-shadow cursor-pointer"

onclick="showLADOnMap('${lad.lad\_code}')">

<div class="flex items-center justify-between">

<div class="flex items-center min-w-0 flex-1">

<span class="w-8 h-8 bg-gradient-to-r from-blue-500 to-indigo-600 text-white rounded-full flex items-center justify-center text-sm font-bold mr-3 flex-shrink-0">

${index + 1}

</span>

<div class="min-w-0 flex-1">

<h4 class="font-semibold text-slate-900 truncate">${lad.lad\_name}</h4>

<p class="text-sm text-slate-600">LAD Code: ${lad.lad\_code}</p>

</div>

</div>

<div class="text-right ml-4 flex-shrink-0">

<div class="text-2xl font-bold text-blue-600">${formatNumber(Math.round(lad.total\_capacity))}</div>

<div class="text-sm text-slate-600">kWe</div>

</div>

</div>

<div class="mt-3 grid grid-cols-4 gap-3 text-sm">

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-green-600">${formatNumber(Math.round(lad.operational\_capacity))}</div>

<div class="text-xs text-slate-600">Operational kWe</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-amber-600">${lad.plant\_count}</div>

<div class="text-xs text-slate-600">Total Plants</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-blue-600">${formatNumber(Math.round(avgPlantSize))}</div>

<div class="text-xs text-slate-600">Avg kWe/Plant</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-purple-600">${percentage.toFixed(1)}%</div>

<div class="text-xs text-slate-600">of UK Total</div>

</div>

</div>

<div class="mt-3 grid grid-cols-3 gap-2 text-xs">

<div class="text-center bg-green-50 rounded p-1">

<div class="font-semibold text-green-700">${lad.operational\_count}</div>

<div class="text-green-600">Operational</div>

</div>

<div class="text-center bg-blue-50 rounded p-1">

<div class="font-semibold text-blue-700">${lad.under\_construction}</div>

<div class="text-blue-600">Under Construction</div>

</div>

<div class="text-center bg-amber-50 rounded p-1">

<div class="font-semibold text-amber-700">${formatNumber(Math.round(lad.planning\_capacity))}</div>

<div class="text-amber-600">Planning kWe</div>

</div>

</div>

<div class="mt-2">

<div class="flex justify-between text-xs text-slate-600 mb-1">

<span>Operational Rate</span>

<span>${operationalPercent.toFixed(1)}%</span>

</div>

<div class="w-full bg-slate-200 rounded-full h-2">

<div class="bg-gradient-to-r from-green-400 to-green-600 h-2 rounded-full transition-all duration-300"

style="width: ${Math.min(100, operationalPercent)}%"></div>

</div>

</div>

</div>

`;

}).join('')}

</div>

</div>

`;

}

function renderTop10LPAsByPlantCount() {

if (!adPlantsData || adPlantsData.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">Loading AD plants data...</div>

<button onclick="loadADPlants().then(() => showTop10Tab('lpas-by-plant-count'))"

class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Reload Plants Data

</button>

</div>

`;

}

// Group plants by LPA

const lpaPlantCounts = {};

adPlantsData.forEach(plant => {

const lpaName = plant['lpa\_name'] || 'Unknown LPA';

if (!lpaPlantCounts[lpaName]) {

lpaPlantCounts[lpaName] = {

name: lpaName,

count: 0,

operational: 0,

planning: 0,

totalCapacity: 0,

totalGWh: 0,

plants: []

};

}

lpaPlantCounts[lpaName].count++;

lpaPlantCounts[lpaName].plants.push(plant);

// Add capacity and GWh

const capacity = parseFloat(plant['Capacity(kWe)']) || 0;

const gwh = parseFloat(plant.GWh) || 0;

lpaPlantCounts[lpaName].totalCapacity += capacity;

lpaPlantCounts[lpaName].totalGWh += gwh;

// Count by status

if (plant.Group && plant.Group.includes('Operational')) {

lpaPlantCounts[lpaName].operational++;

} else if (plant.Group && plant.Group.includes('Planning')) {

lpaPlantCounts[lpaName].planning++;

}

});

// Sort by plant count

const sortedLPAs = Object.values(lpaPlantCounts)

.filter(lpa => lpa.name !== 'Unknown LPA')

.sort((a, b) => b.count - a.count)

.slice(0, 10);

if (sortedLPAs.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">No LPA data found.</div>

<div class="text-xs text-slate-500 mt-2">LPA data requires proper plant location matching</div>

</div>

`;

}

const totalLPAs = Object.keys(lpaPlantCounts).length;

const totalPlants = adPlantsData.length;

const avgPlantsPerLPA = totalPlants / totalLPAs;

return `

<div class="space-y-4">

<div class="bg-gradient-to-r from-violet-50 to-purple-50 rounded-lg p-4">

<h3 class="text-lg font-semibold text-slate-900 mb-2">

🗺️ Top 10 Local Planning Authorities by Plant Count

</h3>

<div class="grid grid-cols-2 md:grid-cols-4 gap-4 text-sm">

<div class="text-center">

<div class="text-2xl font-bold text-violet-600">${totalLPAs}</div>

<div class="text-slate-600">Total LPAs</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-purple-600">${totalPlants}</div>

<div class="text-slate-600">Total Plants</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-blue-600">${avgPlantsPerLPA.toFixed(1)}</div>

<div class="text-slate-600">Avg Plants/LPA</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-green-600">${sortedLPAs[0]?.count || 0}</div>

<div class="text-slate-600">Highest Count</div>

</div>

</div>

</div>

<div class="grid gap-3">

${sortedLPAs.map((lpa, index) => {

const operationalPercent = lpa.count > 0 ? ((lpa.operational / lpa.count) \* 100) : 0;

return `

<div class="bg-white border border-slate-200 rounded-lg p-4 hover:shadow-md transition-shadow">

<div class="flex items-center justify-between">

<div class="flex items-center min-w-0 flex-1">

<span class="w-8 h-8 bg-gradient-to-r from-violet-500 to-purple-600 text-white rounded-full flex items-center justify-center text-sm font-bold mr-3 flex-shrink-0">

${index + 1}

</span>

<div class="min-w-0 flex-1">

<h4 class="font-semibold text-slate-900">${lpa.name}</h4>

<p class="text-sm text-slate-600">Local Planning Authority</p>

</div>

</div>

<div class="text-right ml-4 flex-shrink-0">

<div class="text-2xl font-bold text-violet-600">${lpa.count}</div>

<div class="text-sm text-slate-600">Plants</div>

</div>

</div>

<div class="mt-3 grid grid-cols-4 gap-3 text-sm">

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-green-600">${lpa.operational}</div>

<div class="text-xs text-slate-600">Operational</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-orange-600">${lpa.planning}</div>

<div class="text-xs text-slate-600">Planning</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-blue-600">${formatNumber(Math.round(lpa.totalCapacity))}</div>

<div class="text-xs text-slate-600">Total kWe</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-amber-600">${operationalPercent.toFixed(0)}%</div>

<div class="text-xs text-slate-600">Operational</div>

</div>

</div>

<div class="mt-3 flex flex-wrap gap-1">

${lpa.plants.slice(0, 5).map(plant => `

<span class="px-2 py-1 text-xs bg-slate-100 text-slate-700 rounded">

${(plant['Site name'] || plant['Site Name'] || 'Unknown').split(' ').slice(0, 2).join(' ')}

</span>

`).join('')}

${lpa.plants.length > 5 ? `

<span class="px-2 py-1 text-xs bg-slate-200 text-slate-600 rounded">

+${lpa.plants.length - 5} more

</span>

` : ''}

</div>

</div>

`;

}).join('')}

</div>

</div>

`;

}

function renderTop10OperationalPlantsByGWh() {

if (!adPlantsData || adPlantsData.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">Loading AD plants data...</div>

<button onclick="loadADPlants().then(() => showTop10Tab('plants-operational-gtg'))"

class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Reload Plants Data

</button>

</div>

`;

}

// Filter for GtG + GtG&CHP plants and sort by GWh output

const gtgPlantsByGWh = adPlantsData

.filter(plant => plant.Group === 'Operational: GtG + GtG&CHP' && parseFloat(plant.GWh) > 0)

.sort((a, b) => (parseFloat(b.GWh) || 0) - (parseFloat(a.GWh) || 0))

.slice(0, 10);

if (gtgPlantsByGWh.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">No Operational: GtG + GtG&CHP plants with GWh data found.</div>

<div class="text-xs text-slate-500 mt-2">Note: This ranking focuses on plants with actual GWh output data</div>

</div>

`;

}

const totalGtGGWh = gtgPlantsByGWh.reduce((sum, plant) => sum + (parseFloat(plant.GWh) || 0), 0);

return `

<div class="space-y-4">

<div class="bg-gradient-to-r from-green-50 to-emerald-50 rounded-lg p-4">

<h3 class="text-lg font-semibold text-slate-900 mb-2">

💚 Top 10 Operational: GtG + GtG&CHP Plants by GWh Output

</h3>

<div class="grid grid-cols-2 md:grid-cols-4 gap-4 text-sm">

<div class="text-center">

<div class="text-2xl font-bold text-green-600">${adPlantsData.filter(p => p.Group === 'Operational: GtG + GtG&CHP').length}</div>

<div class="text-slate-600">Total GtG Plants</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-blue-600">${formatNumber(Math.round(totalGtGGWh))}</div>

<div class="text-slate-600">Total GWh Output</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-amber-600">${Math.round(totalGtGGWh / gtgPlantsByGWh.length)}</div>

<div class="text-slate-600">Average GWh</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-purple-600">${gtgPlantsByGWh.length}</div>

<div class="text-slate-600">Plants with Data</div>

</div>

</div>

</div>

<div class="grid gap-3">

${gtgPlantsByGWh.map((plant, index) => {

const gwh = parseFloat(plant.GWh) || 0;

const capacity = parseFloat(plant['Capacity(kWe)']) || 0;

const efficiency = capacity > 0 ? ((gwh \* 1000) / capacity \* 8760) \* 100 : 0; // Capacity factor %

return `

<div class="bg-white border border-slate-200 rounded-lg p-4 hover:shadow-md transition-shadow cursor-pointer"

onclick="selectPlantFromTop10(${plant.Latitude}, ${plant.Longitude}, ${JSON.stringify(plant).replace(/"/g, '&quot;')})">

<div class="flex items-center justify-between">

<div class="flex items-center min-w-0 flex-1">

<span class="w-8 h-8 bg-gradient-to-r from-green-500 to-emerald-600 text-white rounded-full flex items-center justify-center text-sm font-bold mr-3 flex-shrink-0">

${index + 1}

</span>

<div class="min-w-0 flex-1">

<h4 class="font-semibold text-slate-900 truncate">${plant['Site name'] || plant['Site Name'] || 'Unknown Site'}</h4>

<p class="text-sm text-slate-600">${plant.Developer || 'Unknown Developer'}</p>

<p class="text-xs text-slate-500">${plant['lad\_name'] || plant.County || 'Unknown Location'}</p>

</div>

</div>

<div class="text-right ml-4 flex-shrink-0">

<div class="text-2xl font-bold text-green-600">${formatNumber(Math.round(gwh))}</div>

<div class="text-sm text-slate-600">GWh/year</div>

</div>

</div>

<div class="mt-3 grid grid-cols-3 gap-3 text-sm">

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-blue-600">${formatNumber(Math.round(capacity))}</div>

<div class="text-xs text-slate-600">kWe Capacity</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-green-600">${plant.Completion || 'N/A'}</div>

<div class="text-xs text-slate-600">Year Online</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-amber-600">${efficiency.toFixed(1)}%</div>

<div class="text-xs text-slate-600">Efficiency</div>

</div>

</div>

${plant.Feedstock ? `

<div class="mt-2 text-xs text-slate-600">

<strong>Feedstock:</strong> ${plant.Feedstock.length > 60 ? plant.Feedstock.substring(0, 60) + '...' : plant.Feedstock}

</div>

` : ''}

</div>

`;

}).join('')}

</div>

</div>

`;

}

function renderTop10OperationalPlantsByCapacity() {

if (!adPlantsData || adPlantsData.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">Loading AD plants data...</div>

<button onclick="loadADPlants().then(() => showTop10Tab('plants-operational-chp'))"

class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">

Reload Plants Data

</button>

</div>

`;

}

// Filter for CHP plants and sort by kWe capacity

const chpPlantsByCapacity = adPlantsData

.filter(plant => plant.Group === 'Operational: CHP' && parseFloat(plant['Capacity(kWe)']) > 0)

.sort((a, b) => (parseFloat(b['Capacity(kWe)']) || 0) - (parseFloat(a['Capacity(kWe)']) || 0))

.slice(0, 10);

if (chpPlantsByCapacity.length === 0) {

return `

<div class="text-center p-8">

<div class="text-slate-600">No Operational: CHP plants with capacity data found.</div>

<div class="text-xs text-slate-500 mt-2">Note: This ranking focuses on plants with kWe capacity data</div>

</div>

`;

}

const totalChpCapacity = chpPlantsByCapacity.reduce((sum, plant) => sum + (parseFloat(plant['Capacity(kWe)']) || 0), 0);

return `

<div class="space-y-4">

<div class="bg-gradient-to-r from-orange-50 to-red-50 rounded-lg p-4">

<h3 class="text-lg font-semibold text-slate-900 mb-2">

🔥 Top 10 Operational: CHP Plants by kWe Capacity

</h3>

<div class="grid grid-cols-2 md:grid-cols-4 gap-4 text-sm">

<div class="text-center">

<div class="text-2xl font-bold text-orange-600">${adPlantsData.filter(p => p.Group === 'Operational: CHP').length}</div>

<div class="text-slate-600">Total CHP Plants</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-red-600">${formatNumber(Math.round(totalChpCapacity))}</div>

<div class="text-slate-600">Total kWe</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-amber-600">${formatNumber(Math.round(totalChpCapacity / chpPlantsByCapacity.length))}</div>

<div class="text-slate-600">Average kWe</div>

</div>

<div class="text-center">

<div class="text-2xl font-bold text-purple-600">${chpPlantsByCapacity.length}</div>

<div class="text-slate-600">Plants with Data</div>

</div>

</div>

</div>

<div class="grid gap-3">

${chpPlantsByCapacity.map((plant, index) => {

const capacity = parseFloat(plant['Capacity(kWe)']) || 0;

const gwh = parseFloat(plant.GWh) || 0;

const totalFeedstock = plant['Totalfeedstock(tpa)'] ? parseFloat(plant['Totalfeedstock(tpa)'].replace(/,/g, '')) : 0;

return `

<div class="bg-white border border-slate-200 rounded-lg p-4 hover:shadow-md transition-shadow cursor-pointer"

onclick="selectPlantFromTop10(${plant.Latitude}, ${plant.Longitude}, ${JSON.stringify(plant).replace(/"/g, '&quot;')})">

<div class="flex items-center justify-between">

<div class="flex items-center min-w-0 flex-1">

<span class="w-8 h-8 bg-gradient-to-r from-orange-500 to-red-600 text-white rounded-full flex items-center justify-center text-sm font-bold mr-3 flex-shrink-0">

${index + 1}

</span>

<div class="min-w-0 flex-1">

<h4 class="font-semibold text-slate-900 truncate">${plant['Site name'] || plant['Site Name'] || 'Unknown Site'}</h4>

<p class="text-sm text-slate-600">${plant.Developer || 'Unknown Developer'}</p>

<p class="text-xs text-slate-500">${plant['lad\_name'] || plant.County || 'Unknown Location'}</p>

</div>

</div>

<div class="text-right ml-4 flex-shrink-0">

<div class="text-2xl font-bold text-red-600">${formatNumber(Math.round(capacity))}</div>

<div class="text-sm text-slate-600">kWe</div>

</div>

</div>

<div class="mt-3 grid grid-cols-3 gap-3 text-sm">

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-green-600">${formatNumber(Math.round(gwh))}</div>

<div class="text-xs text-slate-600">GWh Output</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-blue-600">${plant.Completion || 'N/A'}</div>

<div class="text-xs text-slate-600">Year Online</div>

</div>

<div class="text-center bg-slate-50 rounded p-2">

<div class="font-semibold text-amber-600">${formatNumber(Math.round(totalFeedstock))}</div>

<div class="text-xs text-slate-600">Feedstock t/yr</div>

</div>

</div>

${plant.Feedstock ? `

<div class="mt-2 text-xs text-slate-600">

<strong>Feedstock:</strong> ${plant.Feedstock.length > 60 ? plant.Feedstock.substring(0, 60) + '...' : plant.Feedstock}

</div>

` : ''}

</div>

`;

}).join('')}

</div>

</div>

`;

}

function selectPlantFromTop10(lat, lng, plantData) {

closeTop10Modal();

// Fly to plant location

map.flyTo({

center: [lng, lat],

zoom: 12,

duration: 1500

});

// Show plant info after animation

setTimeout(() => {

openInfoPanel({

type: 'plant',

plant: plantData

});

}, 1500);

}

// ==================== SEARCH FUNCTIONS ====================

function initializeSearch() {

console.log('🔍 Initializing search function...');

console.log('AD Plants Data Length:', adPlantsData?.length || 0);

console.log('Fuse.js Available:', typeof window.Fuse !== 'undefined');

if (adPlantsData && adPlantsData.length > 0 && window.Fuse) {

try {

searchFuse = new window.Fuse(adPlantsData, {

keys: ['Site name', 'Site Name', 'Developer', 'Postcode', 'Group', 'lad\_name', 'LAD name', 'lpa\_name', 'LPA name', 'County'],

threshold: 0.3,

includeScore: true

});

console.log('✅ Search initialized successfully with', adPlantsData.length, 'plants');

} catch (error) {

console.error('❌ Failed to initialize search:', error);

}

} else {

console.warn('⚠️ Search not initialized - missing data or Fuse.js');

}

}

function handleSearch(query) {

const resultsContainer = document.getElementById('search-results');

if (!searchFuse || query.length < 2) {

resultsContainer.classList.add('hidden');

return;

}

const results = searchFuse.search(query).slice(0, 10);

if (results.length === 0) {

resultsContainer.innerHTML = '<div class="p-3 text-sm text-slate-500">No results found</div>';

} else {

resultsContainer.innerHTML = results.map((result, index) => `

<div class="p-3 hover:bg-slate-50 cursor-pointer border-b border-slate-100 last:border-b-0 search-result"

data-index="${index}">

<div class="font-medium text-slate-900">${result.item['Site name'] || result.item['Site Name'] || 'Unknown Site'}</div>

<div class="text-sm text-slate-600">

${result.item.Developer || 'Unknown Developer'} • ${result.item.Group || result.item.Status || 'Unknown Status'}

</div>

<div class="text-xs text-slate-500">${result.item['lad\_name'] || result.item['LAD name'] || result.item.County || 'Unknown Location'}</div>

</div>

`).map((html, index) => {

const div = document.createElement('div');

div.innerHTML = html;

const element = div.firstChild;

element.addEventListener('click', () => selectSearchResult(results[index]));

return element.outerHTML;

}).join('');

}

resultsContainer.classList.remove('hidden');

}

function selectSearchResult(result) {

const plant = result.item;

const resultsContainer = document.getElementById('search-results');

const searchInput = document.getElementById('search-input');

// Hide results and clear input

resultsContainer.classList.add('hidden');

searchInput.value = '';

// Fly to plant location

map.flyTo({

center: [plant.Longitude, plant.Latitude],

zoom: 12,

duration: 1500

});

// Show plant info after animation

setTimeout(() => {

openInfoPanel({

type: 'plant',

plant

});

}, 1500);

}

// ==================== MAP INITIALIZATION FUNCTIONS ====================

async function initializeMap() {

try {

// Initialize map

map = new maplibregl.Map({

container: 'map',

style: createMapStyle(),

center: MAP\_CONFIG.INITIAL\_CENTER,

zoom: MAP\_CONFIG.INITIAL\_ZOOM,

maxZoom: MAP\_CONFIG.MAX\_ZOOM,

minZoom: MAP\_CONFIG.MIN\_ZOOM

});

// Initialize layer manager

layerManager = new LayerManager(map);

map.on('load', async () => {

try {

// IMPORTANT: Load LPA first (bottom layer), then LAD (middle), AD plants will be top

console.log('Loading LPA boundaries first for proper layer ordering...');

try {

const { feature: lpaFeature } = await loadTopoJSONData(URLS.LPA\_TOPO, 'LPA');

lpaData = lpaFeature;

console.log('LPA boundaries loaded:', lpaData.features.length);

await initializeLPALayer(); // Bottom layer

} catch (error) {

console.warn('Failed to load LPA boundaries:', error);

}

// Load LAD boundaries AFTER LPA for proper stacking

console.log('Loading LAD boundaries...');

const { feature: ladFeature } = await loadTopoJSONData(URLS.LAD\_TOPO, 'LADS');

ladBoundariesData = ladFeature;

console.log('LAD boundaries loaded:', ladBoundariesData.features.length);

// Initialize LAD layer (will be above LPA)

await initializeLADLayer();

// Load manure CSV data for info panels (needed immediately)

console.log('Loading manure data for info panels...');

try {

manureData = await loadCSVData(URLS.MANURE\_LAD\_GROUPS, 'Manure by LAD data');

console.log('LAD Manure CSV data loaded:', manureData.length);

const lpaManureData = await loadCSVData(URLS.MANURE\_LPA\_GROUPS, 'Manure by LPA data');

console.log('LPA Manure CSV data loaded:', lpaManureData.length);

// Store both datasets globally for info panels

window.ladManureData = manureData;

window.lpaManureData = lpaManureData;

console.log('Manure data available for info panels');

} catch (error) {

console.warn('Failed to load manure data:', error);

}

// Initialize UI components without loading data layers

updateLayerControls();

updatePlantsOverview(); // Shows "No data loaded"

setupEventListeners();

setupEventHandlers();

setupProximityAnalysis();

hideLoading();

console.log('✅ Minimal app startup complete');

} catch (error) {

console.error('Failed to initialize application:', error);

hideLoading();

showToast('Failed to load application data. Please refresh the page.', 'error');

}

});

map.on('error', (e) => {

console.error('Map error:', e);

showToast('Map error occurred. Please refresh the page.', 'error');

});

} catch (error) {

console.error('Failed to initialize map:', error);

hideLoading();

showToast('Failed to initialize map. Please check your connection.', 'error');

}

}

async function initializeLADLayer() {

try {

// Add LAD source first

if (!map.getSource('lad-source')) {

map.addSource('lad-source', {

type: 'geojson',

data: ladBoundariesData

});

}

// Add LAD fill layer ABOVE LPA layers (MIDDLE LAYER)

if (!map.getLayer('lad-fill')) {

map.addLayer({

id: 'lad-fill',

source: 'lad-source',

type: 'fill',

paint: {

'fill-color': '#e8d5f2',

'fill-opacity': 0.06

},

layout: {

visibility: 'visible'

}

});

}

// Add LAD boundaries outline ABOVE LPA layers (MIDDLE LAYER)

if (!map.getLayer('lad')) {

map.addLayer({

id: 'lad',

source: 'lad-source',

type: 'line',

paint: {

'line-color': '#7e3aa2',

'line-width': 1.5

},

layout: {

visibility: 'visible'

}

});

}

// Register LAD layers with the LayerManager for toggle functionality

layerManager.registry.set('lad-fill', true);

layerManager.registry.set('lad', true);

layerManager.layers['lad-fill'] = true;

layerManager.layers['lad'] = true;

// LPA layer will be added later if needed

console.log('LAD layers initialized and registered');

// Add comprehensive LAD click handler with AD plant priority check

map.on('click', 'lad-fill', async (e) => {

try {

// Check if an AD plant was clicked at the same location

const features = map.queryRenderedFeatures(e.point);

const adPlantFeatures = features.filter(f => f.layer && f.layer.id && f.layer.id.startsWith('ad-'));

if (adPlantFeatures.length > 0) {

console.log('AD plant detected at LAD click location, skipping LAD handler');

return; // Let the general click handler handle AD plants

}

await handleLADClick(e);

} catch (error) {

console.error('Error in LAD fill click handler:', error);

showToast('Error displaying LAD information', 'error');

}

});

map.on('click', 'lad', async (e) => {

try {

// Check if an AD plant was clicked at the same location

const features = map.queryRenderedFeatures(e.point);

const adPlantFeatures = features.filter(f => f.layer && f.layer.id && f.layer.id.startsWith('ad-'));

if (adPlantFeatures.length > 0) {

console.log('AD plant detected at LAD outline click location, skipping LAD handler');

return; // Let the general click handler handle AD plants

}

await handleLADClick(e);

} catch (error) {

console.error('Error in LAD click handler:', error);

showToast('Error displaying LAD information', 'error');

}

});

map.on('mouseenter', 'lad-fill', () => {

try {

map.getCanvas().style.cursor = 'pointer';

} catch (error) {

console.warn('Error setting cursor on LAD mouseenter:', error);

}

});

map.on('mouseleave', 'lad-fill', () => {

try {

map.getCanvas().style.cursor = '';

} catch (error) {

console.warn('Error resetting cursor on LAD mouseleave:', error);

}

});

// Also add mouse events for the outline layer

map.on('mouseenter', 'lad', () => {

try {

map.getCanvas().style.cursor = 'pointer';

} catch (error) {

console.warn('Error setting cursor on LAD outline mouseenter:', error);

}

});

map.on('mouseleave', 'lad', () => {

try {

map.getCanvas().style.cursor = '';

} catch (error) {

console.warn('Error resetting cursor on LAD outline mouseleave:', error);

}

});

// Add intelligent map click handler that prioritizes AD plants over boundary layers

map.on('click', async (e) => {

console.log('Map clicked at:', e.lngLat);

const features = map.queryRenderedFeatures(e.point);

console.log('Features at click point:', features.length, features.map(f => f.layer.id));

// Check for AD plant features first (highest priority)

const adPlantFeatures = features.filter(f => f.layer && f.layer.id && f.layer.id.startsWith('ad-'));

if (adPlantFeatures.length > 0) {

console.log('AD plant click detected, prioritizing over boundary layers');

const plant = adPlantFeatures[0].properties;

// Trigger plant click handler

openInfoPanel({

type: 'plant',

plant

});

return; // Prevent boundary layer handlers from firing

}

});

console.log('LAD layer initialized successfully');

} catch (error) {

console.error('Failed to initialize LAD layer:', error);

console.error('Error details:', JSON.stringify(error));

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

throw error;

}

}

async function initializeLPALayer() {

try {

// Add LPA source first

if (!map.getSource('lpa-source')) {

map.addSource('lpa-source', {

type: 'geojson',

data: lpaData

});

}

// Add LPA fill layer for better interaction (BOTTOM LAYER)

if (!map.getLayer('lpa-fill')) {

map.addLayer({

id: 'lpa-fill',

source: 'lpa-source',

type: 'fill',

paint: {

'fill-color': '#d2b48c',

'fill-opacity': 0.06

},

layout: {

visibility: 'none' // Start hidden

}

});

}

// Add LPA boundaries outline (BOTTOM LAYER)

if (!map.getLayer('lpa')) {

map.addLayer({

id: 'lpa',

source: 'lpa-source',

type: 'line',

paint: {

'line-color': '#8b5a3c',

'line-width': 1.5

},

layout: {

visibility: 'none' // Start hidden

}

});

}

// Register LPA layers with the LayerManager for toggle functionality

layerManager.registry.set('lpa-fill', true);

layerManager.registry.set('lpa', true);

layerManager.layers['lpa-fill'] = true;

layerManager.layers['lpa'] = true;

console.log('LPA layers initialized and registered');

// Add comprehensive LPA click handler with AD plant priority check

map.on('click', 'lpa-fill', async (e) => {

try {

// Check if an AD plant was clicked at the same location

const features = map.queryRenderedFeatures(e.point);

const adPlantFeatures = features.filter(f => f.layer && f.layer.id && f.layer.id.startsWith('ad-'));

if (adPlantFeatures.length > 0) {

console.log('AD plant detected at LPA click location, skipping LPA handler');

return; // Let the general click handler handle AD plants

}

await handleLPAClick(e);

} catch (error) {

console.error('Error in LPA click handler:', error);

showToast('Error displaying LPA information', 'error');

}

});

map.on('mouseenter', 'lpa-fill', () => {

map.getCanvas().style.cursor = 'pointer';

});

map.on('mouseleave', 'lpa-fill', () => {

map.getCanvas().style.cursor = '';

});

// Also add mouse events for the outline layer

map.on('mouseenter', 'lpa', () => {

map.getCanvas().style.cursor = 'pointer';

});

map.on('mouseleave', 'lpa', () => {

map.getCanvas().style.cursor = '';

});

console.log('LPA layer initialized successfully');

} catch (error) {

console.error('Failed to initialize LPA layer:', error);

console.error('Error details:', JSON.stringify(error));

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

throw error;

}

}

// Removed duplicate initializeMapLayers function - using initializeLADLayer instead

// LPA Click Handler - copy of LAD functionality

async function handleLPAClick(e) {

const lpaProperties = e.features[0].properties;

const lpaName = lpaProperties.LPA23NM || lpaProperties.NAME || lpaProperties.name;

const lpaCode = lpaProperties.LPA23CD || lpaProperties.CODE || lpaProperties.code;

console.log('LPA polygon clicked:', lpaName);

// Load required data if not already loaded

if (adPlantsData.length === 0) {

console.log('Loading AD plants data for LPA analysis...');

adPlantsData = await loadCSVData(URLS.AD\_CSV);

console.log('AD plants data loaded:', adPlantsData.length);

}

if (manureData.length === 0) {

console.log('Loading manure data for LPA analysis...');

manureData = await loadCSVData(URLS.MANURE\_LPA\_GROUPS);

console.log('Manure data loaded:', manureData.length);

}

// Find plants in this LPA using lpa\_code and lpa\_name fields

const plantsInLPA = adPlantsData.filter(plant => {

return plant.lpa\_code === lpaCode || plant.lpa\_name === lpaName;

});

console.log(`Found ${plantsInLPA.length} plants in LPA ${lpaName} using code ${lpaCode}`);

if (plantsInLPA.length > 0) {

console.log('Sample plant data:', plantsInLPA[0]);

}

// Find manure data for this LPA

const manureInLPA = manureData.filter(entry => {

return entry['LPA Code'] === lpaCode || entry['LPA Name'] === lpaName;

});

openInfoPanel({

type: 'lpa',

lpaProperties,

plantsInLPA,

manureInLPA

});

}

async function initializeADPlantsLayers() {

try {

const geojson = createADPlantsGeoJSON(adPlantsData);

// Get unique groups

const groups = [...new Set(adPlantsData.map(plant => plant.Group).filter(Boolean))];

for (const group of groups) {

const layerId = generateLayerId(group);

const color = getADPlantColor(group);

console.log(`Creating AD layer ${layerId} with color ${color} for group: ${group}`);

await layerManager.addLayer({

id: layerId,

source: {

id: 'adPlants',

config: {

type: 'geojson',

data: geojson

}

},

type: 'circle',

filter: ['==', ['get', 'Group'], group],

paint: {

'circle-radius': 8, // Fixed size for all markers

'circle-color': color,

'circle-opacity': 0.9,

'circle-stroke-color': '#ffffff',

'circle-stroke-width': 2

},

layout: { visibility: 'none' }

});

// Add click handler

map.on('click', layerId, (e) => {

openInfoPanel({

type: 'plant',

plant: e.features[0].properties

});

});

map.on('mouseenter', layerId, () => {

map.getCanvas().style.cursor = 'pointer';

});

map.on('mouseleave', layerId, () => {

map.getCanvas().style.cursor = '';

});

}

console.log('AD Plants layers initialized:', groups.length, 'groups');

} catch (error) {

console.error('Failed to initialize AD plants layers:', error);

throw error;

}

}

// Initialize manure volume layers using LAD data and CSV manure data

async function initializeManureVolumeLayers() {

try {

console.log('Initializing manure volume layers...');

if (!manureVolumesData || !manureVolumesData.features) {

console.warn('Missing manure volumes TopJSON data for volume layers');

return;

}

// Process and store enhanced feature data globally

window.processedManureFeatures = manureVolumesData.features.map(feature => {

const props = feature.properties;

// Calculate values for each livestock category

const categoryValues = MANURE\_LAYER\_CONFIGS.map(config => ({

config,

value: parseFloat(props[config.field]) || 0

}));

// Find dominant category (highest value)

const dominantCategory = categoryValues.reduce((max, current) =>

current.value > max.value ? current : max

);

// Calculate total manure volume for this hexagon

const totalVolume = categoryValues.reduce((sum, cat) => sum + cat.value, 0);

return {

...feature,

properties: {

...props,

dominantCategory: dominantCategory.config.id,

dominantCategoryName: dominantCategory.config.name,

dominantValue: dominantCategory.value,

totalVolume: totalVolume,

allCategories: categoryValues

}

};

});

// Calculate max values for opacity scaling

const allValues = window.processedManureFeatures

.map(f => f.properties.totalVolume)

.filter(v => v > 0)

.sort((a, b) => a - b);

const maxVolume = allValues[allValues.length - 1];

const percentile90 = allValues[Math.floor(allValues.length \* 0.9)];

console.log(`Manure volumes range: 0 - ${maxVolume} (90th percentile: ${percentile90})`);

// Store scaling values globally

window.manureVolumeScaling = { maxVolume, percentile90 };

// Create a single dynamic manure volumes layer

await layerManager.addLayer({

id: 'manure-volumes-dynamic',

source: {

id: 'manure-volumes-source',

config: {

type: 'geojson',

data: {

type: 'FeatureCollection',

features: []

}

}

},

type: 'fill',

paint: {

'fill-color': ['get', 'fillColor'],

'fill-opacity': ['get', 'fillOpacity']

},

layout: { visibility: 'none' }

});

// Add click handler for dynamic layer

map.on('click', 'manure-volumes-dynamic', (e) => {

if (e.features && e.features.length > 0) {

const feature = e.features[0];

const configId = feature.properties.dominantCategory;

const config = MANURE\_LAYER\_CONFIGS.find(c => c.id === configId);

if (config) {

showManureVolumePopup(feature.properties, config, e.lngLat);

}

}

});

// Add hover effects

map.on('mouseenter', 'manure-volumes-dynamic', () => {

map.getCanvas().style.cursor = 'pointer';

});

map.on('mouseleave', 'manure-volumes-dynamic', () => {

map.getCanvas().style.cursor = '';

});

// Create individual category layers for toggle functionality

for (const config of MANURE\_LAYER\_CONFIGS) {

await layerManager.addLayer({

id: config.id,

source: {

id: `${config.id}-source`,

config: {

type: 'geojson',

data: {

type: 'FeatureCollection',

features: []

}

}

},

type: 'fill',

paint: {

'fill-color': config.color,

'fill-opacity': 0.7

},

layout: { visibility: 'none' }

});

console.log(`Created layer for ${config.name}`);

}

console.log('Manure volume layers initialized successfully');

} catch (error) {

console.error('Failed to initialize manure volume layers:', error);

throw error;

}

}

// Create heat map GeoJSON from CSV data

function createManureHeatMapFromCSV(boundaries, csvData, config) {

const features = boundaries.features.map(feature => {

const ladName = feature.properties.LAD23NM || feature.properties.NAME;

const ladCode = feature.properties.LAD23CD || feature.properties.CODE;

// Find matching manure data

const manureEntry = csvData.find(entry =>

entry.lad\_name === ladName ||

entry.lad\_code === ladCode ||

(entry.lad\_name && ladName && entry.lad\_name.toLowerCase().includes(ladName.toLowerCase())) ||

(ladName && entry.lad\_name && ladName.toLowerCase().includes(entry.lad\_name.toLowerCase()))

);

let manureValue = 0;

let gwhValue = 0;

if (manureEntry && config.csvField) {

// Extract tonnes value, removing commas and parsing

const tonnesField = config.csvField + '\_tonnes';

const gwhField = config.csvField + '\_gwh';

const rawTonnes = manureEntry[tonnesField];

const rawGwh = manureEntry[gwhField];

if (rawTonnes) {

manureValue = parseFloat(String(rawTonnes).replace(/,/g, '')) || 0;

}

if (rawGwh) {

gwhValue = parseFloat(String(rawGwh).replace(/,/g, '')) || 0;

}

}

return {

...feature,

properties: {

...feature.properties,

manureValue,

gwhValue,

category: config.name,

ladName,

ladCode

}

};

});

return {

type: 'FeatureCollection',

features: features.filter(f => f.properties.manureValue > 0)

};

}

// Show detailed popup for manure volume data from TopJSON hexagonal polygons

function showManureVolumePopup(properties, config, lngLat) {

const dominantValue = properties.dominantValue || 0;

const totalVolume = properties.totalVolume || 0;

const dominantCategoryName = properties.dominantCategoryName || config.name;

const allCategories = properties.allCategories || [];

// Build breakdown of all livestock categories

let categoryBreakdown = '';

if (allCategories.length > 0) {

const sortedCategories = allCategories

.filter(cat => cat.value > 0)

.sort((a, b) => b.value - a.value);

categoryBreakdown = sortedCategories.map(cat => `

<div class="flex justify-between items-center py-1">

<div class="flex items-center">

<div class="w-3 h-3 rounded mr-2" style="background-color: ${cat.config.color}"></div>

<span class="text-xs">${cat.config.name}</span>

</div>

<span class="text-xs font-medium">${formatNumber(Math.round(cat.value))} t/yr</span>

</div>

`).join('');

}

let popupContent = `

<div class="p-4 max-w-sm">

<h4 class="font-bold text-gray-800 mb-3">Manure Production Hexagon</h4>

<div class="mb-3">

<div class="text-sm font-medium text-gray-700 mb-1">Dominant Category</div>

<div class="flex items-center mb-2">

<div class="w-4 h-4 rounded mr-2" style="background-color: ${config.color}"></div>

<span class="font-medium">${dominantCategoryName}</span>

</div>

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

<strong>${formatNumber(Math.round(dominantValue))} tonnes/year</strong>

</div>

</div>

<div class="mb-3">

<div class="text-sm font-medium text-gray-700 mb-1">Total Production</div>

<div class="text-sm text-green-600 font-semibold">

${formatNumber(Math.round(totalVolume))} tonnes/year

</div>

</div>

${categoryBreakdown ? `

<div class="border-t border-gray-200 pt-3">

<div class="text-sm font-medium text-gray-700 mb-2">All Categories</div>

<div class="max-h-32 overflow-y-auto">

${categoryBreakdown}

</div>

</div>

` : ''}

</div>

`;

new maplibregl.Popup({ closeOnClick: true })

.setLngLat(lngLat)

.setHTML(popupContent)

.addTo(map);

}

// Show detailed popup for polygon data (DNO, water companies, environmental layers)

function showPolygonInfoPanel(properties, layerName, color, lngLat) {

const panel = document.getElementById('info-panel');

const title = document.getElementById('info-panel-title');

const content = document.getElementById('info-panel-content');

if (!panel || !title || !content) return;

// Set title

title.textContent = `🗺️ ${layerName}`;

// Build properties list

const propertyEntries = Object.entries(properties)

.filter(([key, value]) => value !== null && value !== undefined && value !== '')

.sort(([a], [b]) => a.localeCompare(b));

let html = `

<div class="space-y-4">

<div class="bg-white border rounded-lg p-4">

<div class="flex items-center mb-3">

<div class="w-4 h-4 rounded mr-2" style="background-color: ${color}"></div>

<h3 class="font-semibold text-gray-800">${layerName}</h3>

</div>

<div class="space-y-2 max-h-96 overflow-y-auto">

`;

if (propertyEntries.length === 0) {

html += '<p class="text-gray-500 text-sm">No additional information available</p>';

} else {

html += propertyEntries.map(([key, value]) => {

// Format key for display

const displayKey = key.replace(/\_/g, ' ')

.replace(/([a-z])([A-Z])/g, '$1 $2')

.replace(/\b\w/g, l => l.toUpperCase());

// Format value

let displayValue = String(value);

if (typeof value === 'number' && value > 1000) {

displayValue = formatNumber(value);

}

return `

<div class="flex justify-between items-start py-2 border-b border-gray-100 last:border-b-0">

<div class="text-sm font-medium text-gray-600 flex-shrink-0 mr-3">

${displayKey}:

</div>

<div class="text-sm text-gray-900 text-right break-words">

${displayValue}

</div>

</div>

`;

}).join('');

}

html += `

</div>

</div>

<div class="bg-gray-50 border rounded-lg p-3">

<button onclick="closeInfoPanel()"

class="w-full bg-gray-600 text-white px-4 py-2 rounded hover:bg-gray-700 transition-colors">

Close

</button>

</div>

</div>

`;

content.innerHTML = html;

// Show the panel

panel.classList.remove('translate-x-full');

panel.classList.add('translate-x-0');

// Also show a small popup at the click location

const popupContent = `

<div class="p-3 max-w-sm">

<h4 class="font-bold text-gray-800 mb-2">${layerName}</h4>

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

Click area details loaded in side panel →

</div>

</div>

`;

new maplibregl.Popup({ closeOnClick: true })

.setLngLat(lngLat)

.setHTML(popupContent)

.addTo(map);

console.log(`${layerName} info panel displayed`);

}

// Update manure volumes display based on active category toggles

// Legacy function - replaced by updateManureDisplay()

function updateManureVolumesDisplay() {

console.log('Legacy updateManureVolumesDisplay called - redirecting to updateManureDisplay');

updateManureDisplay();

}

// Update manure controls UI after loading

// ==================== EVENT LISTENERS ====================

function setupEventListeners() {

// Search functionality

const searchInput = document.getElementById('search-input');

const searchResults = document.getElementById('search-results');

searchInput.addEventListener('input', (e) => {

handleSearch(e.target.value);

});

// Hide search results when clicking outside

document.addEventListener('click', (e) => {

if (!searchInput.contains(e.target) && !searchResults.contains(e.target)) {

searchResults.classList.add('hidden');

}

});

// Map click handler for proximity analysis

map.on('click', (e) => {

if (!proximityAnalysisMode) return;

// Prevent other click handlers from running

e.preventDefault();

performProximityAnalysis(e.lngLat);

});

// Layer checkbox handlers

document.addEventListener('change', async (e) => {

if (e.target.classList.contains('layer-checkbox')) {

const layerId = e.target.dataset.layer;

const isChecked = e.target.checked;

// Load layer data on-demand if checking

if (isChecked) {

try {

// Load AD plants layers

if (layerId.startsWith('ad-') && !adLayersLoaded) {

await loadADPlantsLayers();

}

// Load manure layers - check for any manure layer config IDs

else if (MANURE\_LAYER\_CONFIGS.some(config => config.id === layerId) && !manureLayersLoaded) {

await loadManureLayers();

}

// Load individual environmental layers on demand

else if (layerId === 'aonb' && !layerManager.hasLayer('aonb')) {

await loadAONBLayer();

}

else if (layerId === 'sssi' && !layerManager.hasLayer('sssi')) {

await loadSSSILayer();

}

else if (layerId === 'nvz' && !layerManager.hasLayer('nvz')) {

await loadNVZLayer();

}

else if (layerId === 'flood' && !layerManager.hasLayer('flood')) {

await loadFloodLayer();

}

// Initialize environmental section if needed

else if (['aonb', 'sssi', 'nvz', 'flood'].includes(layerId) && !environmentalLayersLoaded) {

await loadEnvironmentalLayers();

}

// Load infrastructure layers individually on demand

else if (layerId === 'dno' && !layerManager.hasLayer('dno')) {

await loadDNOLayer();

}

else if (layerId === 'water' && !layerManager.hasLayer('water')) {

await loadWaterLayer();

}

else if (layerId === 'brownfield' && !layerManager.hasLayer('brownfield')) {

await loadBrownfieldLayer();

}

// Initialize infrastructure section if needed

else if (['dno', 'water', 'nts', 'brownfield'].includes(layerId) && !infrastructureLayersLoaded) {

await loadInfrastructureLayers();

}

// Load agricultural layers for any ALC grade

else if (layerId.startsWith('alc-grade-') && !agriculturalLayersLoaded) {

await loadAgriculturalLayers();

}

} catch (error) {

console.error(`Failed to load data for layer ${layerId}:`, error);

showToast(`Failed to load ${layerId} layer`, 'error');

e.target.checked = false;

return;

}

}

// Toggle layer visibility

// Special handling for LAD layer - toggle both fill and outline

if (layerId === 'lad') {

try {

map.setLayoutProperty('lad', 'visibility', isChecked ? 'visible' : 'none');

map.setLayoutProperty('lad-fill', 'visibility', isChecked ? 'visible' : 'none');

if (isChecked) {

activeLayers.add('lad');

activeLayers.add('lad-fill');

} else {

activeLayers.delete('lad');

activeLayers.delete('lad-fill');

}

console.log(`LAD layer visibility set to ${isChecked ? 'visible' : 'none'}`);

} catch (error) {

console.error(`Failed to toggle LAD layer:`, error);

e.target.checked = !isChecked;

showToast('Failed to toggle LAD layer', 'error');

}

}

// Special handling for LPA layer

else if (layerId === 'lpa') {

try {

map.setLayoutProperty('lpa-fill', 'visibility', isChecked ? 'visible' : 'none');

map.setLayoutProperty('lpa', 'visibility', isChecked ? 'visible' : 'none');

if (isChecked) {

activeLayers.add('lpa');

activeLayers.add('lpa-fill');

} else {

activeLayers.delete('lpa');

activeLayers.delete('lpa-fill');

}

console.log(`LPA layer visibility set to ${isChecked ? 'visible' : 'none'}`);

} catch (error) {

console.error(`Failed to toggle LPA layer:`, error);

e.target.checked = !isChecked;

showToast('Failed to toggle LPA layer', 'error');

}

}

// Handle manure volume layers with special logic

else if (layerId.startsWith('manure-') && manureVolumesData) {

// Don't handle manure checkboxes here - they use onchange="updateManureDisplay()"

// This prevents conflicts between the two event systems

console.log('Manure checkbox handled by onchange event, skipping layer toggle handler');

return;

}

// Handle environmental layers

else if (['aonb', 'sssi', 'nvz', 'flood'].includes(layerId)) {

try {

if (layerManager && layerManager.layers[layerId]) {

layerManager.toggle(layerId);

if (isChecked) {

activeLayers.add(layerId);

} else {

activeLayers.delete(layerId);

}

console.log(`Environmental layer ${layerId} toggled to ${isChecked}`);

} else {

console.warn(`Environmental layer ${layerId} not found in layer manager`);

e.target.checked = false;

showToast(`${layerId.toUpperCase()} layer not ready`, 'error');

}

} catch (error) {

console.error(`Failed to toggle environmental layer ${layerId}:`, error);

e.target.checked = !isChecked;

showToast(`Failed to toggle ${layerId.toUpperCase()} layer`, 'error');

}

}

// Handle infrastructure layers with proper toggle functionality

else if (['dno', 'water', 'brownfield'].includes(layerId)) {

if (isChecked) {

// Load layer on demand when checkbox is checked

if (layerId === 'dno' && !layerManager.hasLayer('dno')) {

await loadDNOLayer();

} else if (layerId === 'water' && !layerManager.hasLayer('water')) {

await loadWaterLayer();

} else if (layerId === 'brownfield' && !layerManager.hasLayer('brownfield')) {

await loadBrownfieldLayer();

}

// Show the layer

if (layerManager.hasLayer(layerId)) {

layerManager.toggle(layerId, true);

activeLayers.add(layerId);

}

} else {

// Hide the layer when unchecked

if (layerManager.hasLayer(layerId)) {

layerManager.toggle(layerId, false);

activeLayers.delete(layerId);

}

}

}

// Handle other layers with the layer manager

else if (layerManager.layers[layerId]) {

layerManager.toggle(layerId, isChecked);

if (isChecked) {

activeLayers.add(layerId);

} else {

activeLayers.delete(layerId);

}

} else if (isChecked) {

console.warn(`Layer ${layerId} not found in registry`);

e.target.checked = false;

showToast(`Layer ${layerId} not available`, 'warning');

}

updateLegend();

}

});

// Accordion toggles with on-demand loading

document.addEventListener('click', (e) => {

if (e.target.closest('.accordion-trigger')) {

const trigger = e.target.closest('.accordion-trigger');

const content = trigger.nextElementSibling;

const icon = trigger.querySelector('.accordion-icon');

const groupId = trigger.dataset.group;

if (content.classList.contains('collapsed')) {

content.classList.remove('collapsed');

content.classList.add('expanded');

if (icon) {

icon.style.transform = 'rotate(180deg)';

}

// Load data on demand

if (groupId === 'ad-plants' && !adLayersLoaded) {

content.innerHTML = '<div class="p-4 text-sm text-slate-500">Loading AD Plants data...</div>';

loadADPlantsLayers();

} else if (groupId === 'manure' && !manureLayersLoaded) {

content.innerHTML = '<div class="p-4 text-sm text-slate-500">Loading manure data...</div>';

loadManureLayers();

} else if (groupId === 'infrastructure' && !infrastructureLayersLoaded) {

content.innerHTML = '<div class="p-4 text-sm text-slate-500">Loading infrastructure data...</div>';

loadInfrastructureLayers().then(() => {

updateLayerControls();

// NTS is loaded but checkbox remains unchecked - user must manually toggle

});

}

} else {

content.classList.remove('expanded');

content.classList.add('collapsed');

if (icon) {

icon.style.transform = 'rotate(0deg)';

}

}

}

});

// Button handlers

document.getElementById('uncheck-all').addEventListener('click', () => {

layerManager.uncheckAllLayers();

activeLayers.clear();

activeLayers.add('lad');

// Update checkboxes

document.querySelectorAll('.layer-checkbox').forEach(checkbox => {

checkbox.checked = checkbox.dataset.layer === 'lad';

});

updateLegend();

showToast('All layers unchecked', 'info');

});

document.getElementById('show-top10').addEventListener('click', openTop10Modal);

document.getElementById('close-top10-modal').addEventListener('click', closeTop10Modal);

document.getElementById('modal-backdrop').addEventListener('click', closeTop10Modal);

document.getElementById('reset-view').addEventListener('click', () => {

map.flyTo({

center: MAP\_CONFIG.INITIAL\_CENTER,

zoom: MAP\_CONFIG.INITIAL\_ZOOM,

duration: 1500

});

});

document.getElementById('close-info-panel').addEventListener('click', closeInfoPanel);

// Export functionality event handlers

const exportToggle = document.getElementById('export-toggle');

if (exportToggle) {

exportToggle.addEventListener('click', toggleExportPanel);

document.getElementById('export-lad-manure')?.addEventListener('click', () => exportData('lad-manure'));

document.getElementById('export-lpa-manure')?.addEventListener('click', () => exportData('lpa-manure'));

document.getElementById('export-heatmap-data')?.addEventListener('click', exportHeatMapData);

document.getElementById('export-ad-plants')?.addEventListener('click', () => exportData('ad-plants'));

}

// Top 10 tab handlers

document.querySelectorAll('.top10-tab').forEach(tab => {

tab.addEventListener('click', (e) => {

showTop10Tab(e.target.dataset.tab);

});

});

// Sidebar collapse (mobile)

document.getElementById('toggle-sidebar').addEventListener('click', () => {

const sidebar = document.getElementById('sidebar');

sidebar.classList.toggle('open');

});

// AD panel toggle

document.getElementById('toggle-ad-panel').addEventListener('click', (e) => {

e.stopPropagation();

const panel = document.getElementById('plants-overview');

const button = document.getElementById('toggle-ad-panel');

const icon = button.querySelector('svg');

if (panel.style.display === 'none') {

panel.style.display = 'block';

icon.style.transform = 'rotate(0deg)';

} else {

panel.style.display = 'none';

icon.style.transform = 'rotate(-90deg)';

}

});

}

// ==================== APPLICATION INITIALIZATION ====================

// Initialize application when DOM is ready

document.addEventListener('DOMContentLoaded', () => {

console.log('Initializing UK Anaerobic Digestion Mapping Application...');

// Check for required dependencies

if (typeof maplibregl === 'undefined') {

console.error('MapLibre GL JS not loaded');

showToast('MapLibre GL JS failed to load. Please refresh the page.', 'error');

return;

}

if (typeof Papa === 'undefined') {

console.error('Papa Parse not loaded');

showToast('Papa Parse failed to load. Please refresh the page.', 'error');

return;

}

if (typeof topojson === 'undefined') {

console.error('TopoJSON not loaded');

showToast('TopoJSON failed to load. Please refresh the page.', 'error');

return;

}

console.log('All dependencies loaded successfully');

initializeMap();

});

// Handle window resize

window.addEventListener('resize', () => {

if (map) {

map.resize();

}

});

// ==================== EXPORT FUNCTIONALITY ====================

function toggleExportPanel() {

const panel = document.getElementById('export-panel');

const isHidden = panel.classList.contains('hidden');

if (isHidden) {

panel.classList.remove('hidden');

updateExportLayerOptions();

} else {

panel.classList.add('hidden');

}

}

function updateExportLayerOptions() {

const select = document.getElementById('export-layer-select');

if (!select) return;

// Clear existing options except the first one

select.innerHTML = '<option value="">Select Layer</option>';

// Add manure layers that are currently loaded

MANURE\_LAYER\_CONFIGS.forEach(config => {

if (layerManager && layerManager.registry.has(config.id)) {

const option = document.createElement('option');

option.value = config.id;

option.textContent = config.name;

select.appendChild(option);

}

});

}

async function exportData(type) {

try {

showToast('Preparing export...', 'info');

let data, filename, csvContent;

switch (type) {

case 'lad-manure':

data = window.ladManureData;

filename = 'lad\_manure\_data.csv';

csvContent = convertToCSV(data);

break;

case 'lpa-manure':

data = window.lpaManureData;

filename = 'lpa\_manure\_data.csv';

csvContent = convertToCSV(data);

break;

case 'ad-plants':

data = adPlantsData;

filename = 'ad\_plants\_data.csv';

csvContent = convertToCSV(data);

break;

default:

throw new Error('Unknown export type');

}

if (!data || data.length === 0) {

showToast('No data available for export', 'error');

return;

}

downloadFile(csvContent, filename, 'text/csv');

showToast(`Exported ${data.length} records`, 'success');

} catch (error) {

console.error('Export error:', error);

showToast('Export failed: ' + error.message, 'error');

}

}

function exportHeatMapData() {

const select = document.getElementById('export-layer-select');

const selectedLayer = select.value;

if (!selectedLayer) {

showToast('Please select a layer to export', 'error');

return;

}

try {

const layerConfig = MANURE\_LAYER\_CONFIGS.find(config => config.id === selectedLayer);

if (!layerConfig) {

throw new Error('Layer configuration not found');

}

// Create GeoJSON from the heat map data

const heatMapData = createManureHeatMapFromCSV(ladBoundaries, window.ladManureData, layerConfig);

const geoJsonContent = JSON.stringify(heatMapData, null, 2);

const filename = `${selectedLayer}\_heatmap.geojson`;

downloadFile(geoJsonContent, filename, 'application/geo+json');

showToast(`Exported ${layerConfig.name} heat map`, 'success');

} catch (error) {

console.error('Heat map export error:', error);

showToast('Heat map export failed: ' + error.message, 'error');

}

}

function convertToCSV(data) {

if (!data || data.length === 0) {

return '';

}

// Get headers from the first row

const headers = Object.keys(data[0]);

// Create CSV content

const csvRows = [headers.join(',')];

for (const row of data) {

const values = headers.map(header => {

const value = row[header];

// Handle values that contain commas, quotes, or newlines

if (typeof value === 'string' && (value.includes(',') || value.includes('"') || value.includes('\n'))) {

return `"${value.replace(/"/g, '""')}"`;

}

return value;

});

csvRows.push(values.join(','));

}

return csvRows.join('\n');

}

function downloadFile(content, filename, mimeType) {

const blob = new Blob([content], { type: mimeType });

const url = URL.createObjectURL(blob);

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

link.href = url;

link.download = filename;

link.style.display = 'none';

document.body.appendChild(link);

link.click();

document.body.removeChild(link);

// Clean up the URL object

setTimeout(() => URL.revokeObjectURL(url), 100);

}

// Export functions for global access

window.selectPlantFromTop10 = selectPlantFromTop10;

// UI Helper Functions to prevent null reference errors

function toggleNextElement(button) {

try {

const nextElement = button.nextElementSibling;

if (nextElement && nextElement.style) {

nextElement.style.display = nextElement.style.display === 'none' ? 'block' : 'none';

}

} catch (error) {

console.warn('Error toggling next element:', error);

}

}

function toggleManureContent(button) {

try {

const contentElement = button.closest('.bg-white')?.querySelector('.manure-content');

if (contentElement && contentElement.style) {

contentElement.style.display = contentElement.style.display === 'none' ? 'block' : 'none';

}

} catch (error) {

console.warn('Error toggling manure content:', error);

}

}

// ==================== PROXIMITY ANALYSIS ====================

function setupProximityAnalysis() {

const proximityToggle = document.getElementById('proximity-toggle');

const proximityInstructions = document.getElementById('proximity-instructions');

if (!proximityToggle) return;

proximityToggle.addEventListener('click', () => {

proximityAnalysisMode = !proximityAnalysisMode;

if (proximityAnalysisMode) {

proximityToggle.textContent = 'Exit Proximity Analysis';

proximityToggle.classList.remove('bg-purple-600', 'hover:bg-purple-700');

proximityToggle.classList.add('bg-red-600', 'hover:bg-red-700');

proximityInstructions.classList.remove('hidden');

map.getCanvas().style.cursor = 'crosshair';

} else {

proximityToggle.textContent = 'Proximity Analysis';

proximityToggle.classList.remove('bg-red-600', 'hover:bg-red-700');

proximityToggle.classList.add('bg-purple-600', 'hover:bg-purple-700');

proximityInstructions.classList.add('hidden');

map.getCanvas().style.cursor = '';

// Remove proximity marker if exists

if (proximityMarker) {

proximityMarker.remove();

proximityMarker = null;

}

}

});

}

async function performProximityAnalysis(lngLat) {

try {

console.log('Starting proximity analysis at:', lngLat);

// Remove proximity marker if exists (user doesn't want bubble)

if (proximityMarker) {

proximityMarker.remove();

proximityMarker = null;

}

// Ensure data is loaded

await Promise.all([

loadADPlantsDataIfNeeded(),

loadManureDataIfNeeded(),

loadManureVolumesDataIfNeeded()

]);

// Perform analysis

const analysis = await analyzeProximity(lngLat);

// Display results

showProximityResults(lngLat, analysis);

} catch (error) {

console.error('Error performing proximity analysis:', error);

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

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

showToast('Proximity analysis failed: ' + (error.message || 'Unknown error'), 'error');

}

}

async function analyzeProximity(lngLat) {

const distances = [10, 25, 50, 100]; // miles

const results = {

location: lngLat,

adPlants: {},

manureVolumes: {},

environmental: {}

};

// 1. Analyze AD Plants by distance and group (operational only)

const operationalGroups = [

'Operational: GtG + GtG&CHP',

'Operational: CHP',

'Operational: Other'

];

for (const distance of distances) {

const plantsInRadius = findPlantsWithinRadius(lngLat, distance);

// Filter and group only operational plants

const operationalPlantsByGroup = {};

let totalOperational = 0;

plantsInRadius.forEach(plant => {

const group = plant.Group || 'Other';

if (operationalGroups.includes(group)) {

if (!operationalPlantsByGroup[group]) {

operationalPlantsByGroup[group] = [];

}

operationalPlantsByGroup[group].push(plant);

totalOperational++;

}

});

results.adPlants[`${distance}miles`] = {

total: totalOperational,

byGroup: operationalPlantsByGroup

};

}

// 2. Analyze Manure Volumes by distance and type

for (const distance of distances) {

const manureInRadius = findManureWithinRadius(lngLat, distance);

results.manureVolumes[`${distance}miles`] = manureInRadius;

}

// 3. Calculate distances to nearest environmental features

results.environmental = await calculateEnvironmentalDistances(lngLat);

return results;

}

function findPlantsWithinRadius(center, radiusMiles) {

if (!adPlantsData || adPlantsData.length === 0) return [];

const radiusKm = radiusMiles \* 1.60934; // Convert miles to kilometers

return adPlantsData.filter(plant => {

if (!plant.Latitude || !plant.Longitude) return false;

const distance = calculateDistance(

center.lat, center.lng,

plant.Latitude, plant.Longitude

);

return distance <= radiusKm;

});

}

function findManureWithinRadius(center, radiusMiles) {

if (!manureVolumesData || !manureVolumesData.features) return {};

const radiusKm = radiusMiles \* 1.60934;

const manureTypes = ['Beef FYM', 'Beef Slurry', 'Dairy FYM', 'Dairy Slurry', 'Broilers', 'Layers', 'Pigs', 'Sheep'];

const result = {};

manureTypes.forEach(type => {

let totalVolume = 0;

let hexagonCount = 0;

manureVolumesData.features.forEach(feature => {

// Get centroid of hexagon

const coords = feature.geometry.coordinates[0][0];

if (!coords || coords.length < 2) return;

const hexLng = coords[0];

const hexLat = coords[1];

const distance = calculateDistance(center.lat, center.lng, hexLat, hexLng);

if (distance <= radiusKm) {

const volumeResult = aggregateManureVolumes(feature.properties, type);

if (volumeResult.total > 0) {

totalVolume += volumeResult.total;

hexagonCount++;

}

}

});

result[type] = {

totalKg: totalVolume,

totalTonnes: Math.round(totalVolume / 1000),

hexagonCount: hexagonCount

};

});

return result;

}

// REMOVED - This function was causing crashes due to undefined ENVIRONMENTAL\_URLS

// Environmental distance calculations now use the map layers directly

async function calculateEnvironmentalDistances(lngLat) {

const distances = {};

// Create analysis point using turf.js (only if turf is available)

let analysisPoint = null;

try {

analysisPoint = turf.point([lngLat.lng, lngLat.lat]);

} catch (e) {

console.warn('Turf.js not available for advanced distance calculations');

}

// Calculate distance to nearest AONB

if (map.getLayer('aonb') && map.getSource('aonb')) {

try {

const aonbSource = map.getSource('aonb');

if (aonbSource.\_data && aonbSource.\_data.features) {

let minDistance = Infinity;

let nearestFeature = null;

for (const feature of aonbSource.\_data.features) {

try {

if (analysisPoint) {

const distance = turf.pointToPolygon(analysisPoint, feature, { units: 'miles' });

if (distance < minDistance) {

minDistance = distance;

nearestFeature = feature;

}

} else {

// Fallback to simple distance calculation

const centroid = turf.centroid(feature);

const distance = turf.distance([lngLat.lng, lngLat.lat], centroid.geometry.coordinates, { units: 'miles' });

if (distance < minDistance) {

minDistance = distance;

nearestFeature = feature;

}

}

} catch (e) {

continue;

}

}

distances.AONB = nearestFeature ? {

name: nearestFeature.properties.name || nearestFeature.properties.AONB\_NAME || 'AONB',

distance: Math.round(minDistance \* 10) / 10

} : { name: 'No AONB found', distance: null };

} else {

distances.AONB = { name: 'AONB data not available', distance: null };

}

} catch (error) {

distances.AONB = { name: 'Enable AONB layer for distance', distance: null };

}

} else {

distances.AONB = { name: 'Enable AONB layer for distance', distance: null };

}

// Calculate distance to nearest SSSI

if (map.getLayer('sssi') && map.getSource('sssi')) {

try {

const sssiSource = map.getSource('sssi');

if (sssiSource.\_data && sssiSource.\_data.features) {

let minDistance = Infinity;

let nearestFeature = null;

for (const feature of sssiSource.\_data.features) {

try {

if (analysisPoint) {

const distance = turf.pointToPolygon(analysisPoint, feature, { units: 'miles' });

if (distance < minDistance) {

minDistance = distance;

nearestFeature = feature;

}

} else {

// Fallback to simple distance calculation

const centroid = turf.centroid(feature);

const distance = turf.distance([lngLat.lng, lngLat.lat], centroid.geometry.coordinates, { units: 'miles' });

if (distance < minDistance) {

minDistance = distance;

nearestFeature = feature;

}

}

} catch (e) {

continue;

}

}

distances.SSSI = nearestFeature ? {

name: nearestFeature.properties.SSSI\_NAME || nearestFeature.properties.name || 'SSSI',

distance: Math.round(minDistance \* 10) / 10

} : { name: 'No SSSI found', distance: null };

} else {

distances.SSSI = { name: 'SSSI data not available', distance: null };

}

} catch (error) {

distances.SSSI = { name: 'Enable SSSI layer for distance', distance: null };

}

} else {

distances.SSSI = { name: 'Enable SSSI layer for distance', distance: null };

}

// Calculate distance to nearest NVZ

if (map.getLayer('nvz') && map.getSource('nvz')) {

try {

const nvzSource = map.getSource('nvz');

if (nvzSource.\_data && nvzSource.\_data.features) {

let minDistance = Infinity;

let nearestFeature = null;

for (const feature of nvzSource.\_data.features) {

try {

if (analysisPoint) {

const distance = turf.pointToPolygon(analysisPoint, feature, { units: 'miles' });

if (distance < minDistance) {

minDistance = distance;

nearestFeature = feature;

}

} else {

// Fallback to simple distance calculation

const centroid = turf.centroid(feature);

const distance = turf.distance([lngLat.lng, lngLat.lat], centroid.geometry.coordinates, { units: 'miles' });

if (distance < minDistance) {

minDistance = distance;

nearestFeature = feature;

}

}

} catch (e) {

continue;

}

}

distances.NVZ = nearestFeature ? {

name: nearestFeature.properties.NVZ\_NAME || nearestFeature.properties.name || 'NVZ',

distance: Math.round(minDistance \* 10) / 10

} : { name: 'No NVZ found', distance: null };

} else {

distances.NVZ = { name: 'NVZ data not available', distance: null };

}

} catch (error) {

distances.NVZ = { name: 'Enable NVZ layer for distance', distance: null };

}

} else {

distances.NVZ = { name: 'Enable NVZ layer for distance', distance: null };

}

// Calculate distance to nearest NTS pipeline

if (map.getLayer('nts') && map.getSource('nts')) {

try {

const ntsSource = map.getSource('nts');

if (ntsSource.\_data && ntsSource.\_data.features) {

let minDistance = Infinity;

let nearestFeature = null;

for (const feature of ntsSource.\_data.features) {

try {

if (analysisPoint) {

// Use pointToLineDistance for linear features

const distance = turf.pointToLineDistance(analysisPoint, feature, { units: 'miles' });

if (distance < minDistance) {

minDistance = distance;

nearestFeature = feature;

}

} else {

// Simple fallback for linear features

const coords = feature.geometry.coordinates[0];

const distance = turf.distance([lngLat.lng, lngLat.lat], coords, { units: 'miles' });

if (distance < minDistance) {

minDistance = distance;

nearestFeature = feature;

}

}

} catch (e) {

continue;

}

}

distances.NTS = nearestFeature ? {

name: nearestFeature.properties.name || nearestFeature.properties.GRID\_ID || 'NTS Pipeline',

distance: Math.round(minDistance \* 10) / 10

} : { name: 'No NTS found', distance: null };

} else {

distances.NTS = { name: 'NTS data not available', distance: null };

}

} catch (error) {

distances.NTS = { name: 'NTS calculation error', distance: null };

}

} else {

distances.NTS = { name: 'Load NTS layer for distance', distance: null };

}

return distances;

}

function findNearestFeature(lngLat, features) {

let nearest = null;

let minDistance = Infinity;

features.forEach(feature => {

// Get a representative point from the feature

let featurePoint;

if (feature.geometry.type === 'Point') {

featurePoint = {

lng: feature.geometry.coordinates[0],

lat: feature.geometry.coordinates[1]

};

} else if (feature.geometry.type === 'Polygon' || feature.geometry.type === 'MultiPolygon') {

// Use first coordinate as approximation (could be improved with centroid calculation)

const coords = feature.geometry.type === 'Polygon'

? feature.geometry.coordinates[0][0]

: feature.geometry.coordinates[0][0][0];

featurePoint = {

lng: coords[0],

lat: coords[1]

};

}

if (featurePoint) {

const distance = calculateDistance(lngLat.lat, lngLat.lng, featurePoint.lat, featurePoint.lng);

if (distance < minDistance) {

minDistance = distance;

nearest = { feature, distance };

}

}

});

return nearest;

}

function calculateDistance(lat1, lng1, lat2, lng2) {

const R = 6371; // Earth's radius in kilometers

const dLat = (lat2 - lat1) \* Math.PI / 180;

const dLng = (lng2 - lng1) \* Math.PI / 180;

const a = Math.sin(dLat/2) \* Math.sin(dLat/2) +

Math.cos(lat1 \* Math.PI / 180) \* Math.cos(lat2 \* Math.PI / 180) \*

Math.sin(dLng/2) \* Math.sin(dLng/2);

const c = 2 \* Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

return R \* c;

}

function showProximityResults(location, analysis) {

const infoPanel = document.getElementById('info-panel');

const infoPanelTitle = document.getElementById('info-panel-title');

const infoPanelContent = document.getElementById('info-panel-content');

if (!infoPanel || !infoPanelTitle || !infoPanelContent) return;

infoPanelTitle.textContent = 'Proximity Analysis Results';

let html = `

<div class="space-y-6">

<div class="bg-purple-50 p-3 rounded">

<p class="text-sm text-purple-700">

<strong>Analysis Point:</strong><br>

${location.lat.toFixed(6)}, ${location.lng.toFixed(6)}

</p>

</div>

<div>

<h4 class="font-semibold text-gray-900 mb-3">AD Plants by Distance</h4>

<div class="space-y-3">

`;

// AD Plants section

[10, 25, 50, 100].forEach(distance => {

const data = analysis.adPlants[`${distance}miles`];

html += `

<div class="border-l-4 border-blue-500 pl-3 mb-3">

<div class="cursor-pointer" onclick="toggleProximitySection('plants-${distance}')">

<p class="font-medium text-sm flex items-center">

<span class="mr-2">▶</span>

Within ${distance} miles: ${data.total} operational plants

</p>

</div>

<div id="plants-${distance}" class="mt-2 ml-4 space-y-1" style="display: none;">

`;

Object.keys(data.byGroup).forEach(group => {

const plants = data.byGroup[group];

const count = plants.length;

html += `

<div class="mb-2">

<div class="cursor-pointer text-xs font-medium text-gray-700 mb-1"

onclick="toggleProximitySection('group-${distance}-${group.replace(/[^a-zA-Z0-9]/g, '')}')">

<span class="mr-1">▶</span> ${group}: ${count} plants

</div>

<div id="group-${distance}-${group.replace(/[^a-zA-Z0-9]/g, '')}" class="ml-4 space-y-1" style="display: none;">

`;

plants.forEach(plant => {

const plantName = plant['Site name'] || plant['Site Name'] || plant.Name || plant['Plant Name'] || plant.name || 'Unknown Plant';

const plantDistance = calculateDistance(location.lat, location.lng, plant.Latitude, plant.Longitude);

// For GtG plants, show GWh; for others show kWe capacity

let capacityText;

const isGtGPlant = group && group.includes('GtG');

if (isGtGPlant) {

const gwh = parseFloat(plant.GWh) || 0;

capacityText = gwh > 0 ? `${gwh.toFixed(1)}GWh/yr` : 'GWh N/A';

} else {

const plantCapacity = parseFloat(String(plant['Capacity(kWe)']).replace(/,/g, '')) || 0;

capacityText = plantCapacity > 1000 ? `${(plantCapacity/1000).toFixed(1)}MW` : `${Math.round(plantCapacity)}kW`;

}

// Convert km to miles

const distanceMiles = plantDistance \* 0.621371;

const distanceText = distanceMiles < 1 ? `${Math.round(distanceMiles \* 5280)}ft` : `${distanceMiles.toFixed(1)}mi`;

html += `

<div class="text-xs text-blue-600 hover:text-blue-800 cursor-pointer underline flex justify-between"

onclick="flyToPlant(${plant.Latitude}, ${plant.Longitude}, '${plantName.replace(/'/g, "\\'")}')">

<span>• ${plantName}</span>

<span class="text-gray-500 ml-2">${distanceText} • ${capacityText}</span>

</div>

`;

});

html += `</div></div>`;

});

html += `</div></div>`;

});

html += `

</div>

</div>

<div>

<h4 class="font-semibold text-gray-900 mb-3">Manure Volumes by Distance</h4>

<div class="space-y-3">

`;

// Manure volumes section

[10, 25, 50, 100].forEach(distance => {

const data = analysis.manureVolumes[`${distance}miles`];

const totalAreas = Object.values(data).reduce((sum, vol) => sum + vol.hexagonCount, 0);

const totalTonnes = Object.values(data).reduce((sum, vol) => sum + vol.totalTonnes, 0);

html += `

<div class="border-l-4 border-green-500 pl-3 mb-3">

<div class="cursor-pointer" onclick="toggleProximitySection('manure-${distance}')">

<p class="font-medium text-sm flex items-center">

<span class="mr-2">▶</span>

Within ${distance} miles: ${totalTonnes.toLocaleString()} tonnes (${totalAreas} areas)

</p>

</div>

<div id="manure-${distance}" class="mt-2 ml-4 space-y-1" style="display: none;">

`;

Object.keys(data).forEach(type => {

const volumes = data[type];

if (volumes.totalTonnes > 0) {

html += `<p class="text-xs text-gray-600">• ${type}: ${volumes.totalTonnes.toLocaleString()} tonnes (${volumes.hexagonCount} areas)</p>`;

}

});

html += `</div></div>`;

});

html += `

</div>

</div>

<div>

<h4 class="font-semibold text-gray-900 mb-3">Distance to Environmental Features</h4>

<div class="space-y-2">

`;

// Environmental distances section

if (analysis.environmental) {

const envFeatures = ['AONB', 'SSSI', 'NVZ', 'NTS'];

envFeatures.forEach(feature => {

const data = analysis.environmental[feature];

if (data) {

const distanceText = data.distance !== null

? `${(data.distance \* 0.621371).toFixed(1)} miles`

: 'N/A';

html += `

<div class="border-l-4 border-purple-500 pl-3 py-2">

<p class="text-sm">

<span class="font-medium">${feature}:</span>

<span class="text-gray-600">${distanceText}</span>

</p>

<p class="text-xs text-gray-500">${data.name}</p>

</div>

`;

}

});

}

html += `

</div>

</div>

</div>

`;

infoPanelContent.innerHTML = html;

infoPanel.classList.add('open');

infoPanel.style.transform = 'translateX(0)';

}

function toggleProximitySection(sectionId) {

const section = document.getElementById(sectionId);

const toggle = section?.previousElementSibling?.querySelector('span');

if (section && toggle) {

const isHidden = section.style.display === 'none';

section.style.display = isHidden ? 'block' : 'none';

toggle.textContent = isHidden ? '▼' : '▶';

}

}

function flyToPlant(lat, lng, plantName) {

if (map && lat && lng) {

console.log(`Flying to plant: ${plantName} at ${lat}, ${lng}`);

map.flyTo({

center: [lng, lat],

zoom: 12,

duration: 2000

});

// Create a temporary marker for the plant

const plantMarker = new maplibregl.Marker({

color: '#0f5132'

})

.setLngLat([lng, lat])

.addTo(map);

// Remove the marker after 5 seconds

setTimeout(() => {

plantMarker.remove();

}, 5000);

showToast(`Flying to ${plantName}`, 'success');

}

}