**Practical9 Multi-layer visualization with Filters**

**In this practical, you will learn something about multi-layer visualization using your own json/geojson data by the most simple map api(Openlayers).**

***Before you start, you should have basic knowledge about webmap development. and client side basic knowledge(at least LIVE SERVER in your VSCODE).***

**Step 0 — Project Skeleton**

You do:

• Create a folder (e.g., map-dashboard/) with three files: index.html, styles.css, script.js.

• Put your GeoJSON files in the same folder (e.g., gz\_2010\_us\_050\_00\_500k.json, data.geojson). (*these data are given on canvas*)

You see:

• An empty project that mirrors production structure (HTML + CSS + JS + data).

Core snippet (file list):

图形用户界面, 文本, 应用程序

中度可信度描述已自动生成

Why do this:

• Keeping files side-by-side avoids CORS and path confusion during local development.

**Step 1 — Create the Three‑Column Shell (HTML)**

**When create a website, the first thing you need to do is to create basic layout. (only consider html element)**

You do:

• Add the header and three logical regions: Filters (left), Map (center), Inspector (right).

You see:

• A clear scaffold even before styles: one header and three boxes ready to hold content.

Core snippet (index.html):

<header class="app-header"> <h1>Interactive Map Dashboard</h1> </header> <div class="app-shell"> <aside id="filters">Filters</aside> <main class="map-wrap"><div id="map"></div></main> <aside id="inspector"> <div>Lon, Lat: <span id="lonlat">—</span></div> <div id="feature-info">Click a feature to view attributes.</div> </aside> </div>

**Step 2 — Make It a 3‑Column Grid (CSS)**

**When you have consider well about your html, then use CSS to make your web beautiful.**

You do:

• Convert the shell into a responsive grid; ensure the map div fills its container.

You see:

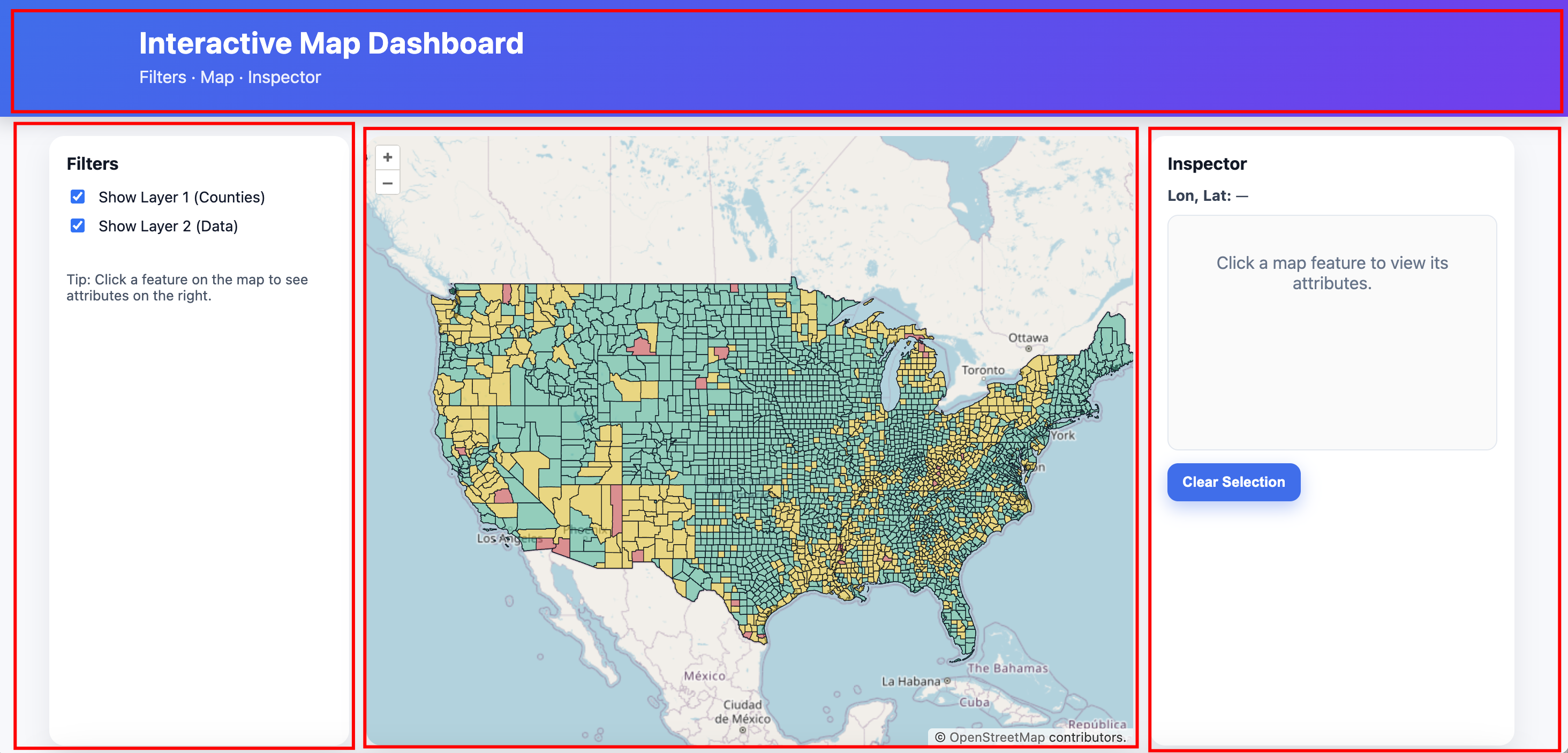
• A stable 3‑column layout on desktop; collapses to a single column on narrow screens.

Core snippet (styles.css):

.app-shell { display: grid; grid-template-columns: 280px 1fr 340px; gap: 16px; } .map-wrap { height: 560px; } .map { width: 100%; height: 100%; } @media (max-width:1100px){ .app-shell { grid-template-columns: 1fr; } }

Why do this:

• **Grid** lets you reason in columns. **Fixing .map height** prevents collapsing maps and ensures tiles paint reliably.



**Step 3 — Load OpenLayers and Add a Base Map**

**Connect to webmap API (HTML + JavaScript)**

You do:

• Add the OpenLayers CSS/JS to index.html; initialize a map with an OSM base layer in script.js.

You see:

• A working basemap centered on the USA (or your chosen extent), confirming OL loaded correctly.

Core snippet (index.html head & end of body):

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/ol@latest/ol.css" /> ... <script src="https://cdn.jsdelivr.net/npm/ol@latest/dist/ol.js"></script> <script src="script.js"></script>

Core snippet (script.js):

const map = new ol.Map({ target: 'map', layers: [ new ol.layer.Tile({ source: new ol.source.OSM() }) ], view: new ol.View({ center: ol.proj.fromLonLat([-98.5795,39.8283]), zoom: 4 }) });

**Step 4 — Add Your Data Layers (Vector)**

**Visualize your own data in mainstream data format via JSON/GeoJSON**

You do:

• Load two vector layers(one from JSON, another from local GeoJSON) and add them to the map.

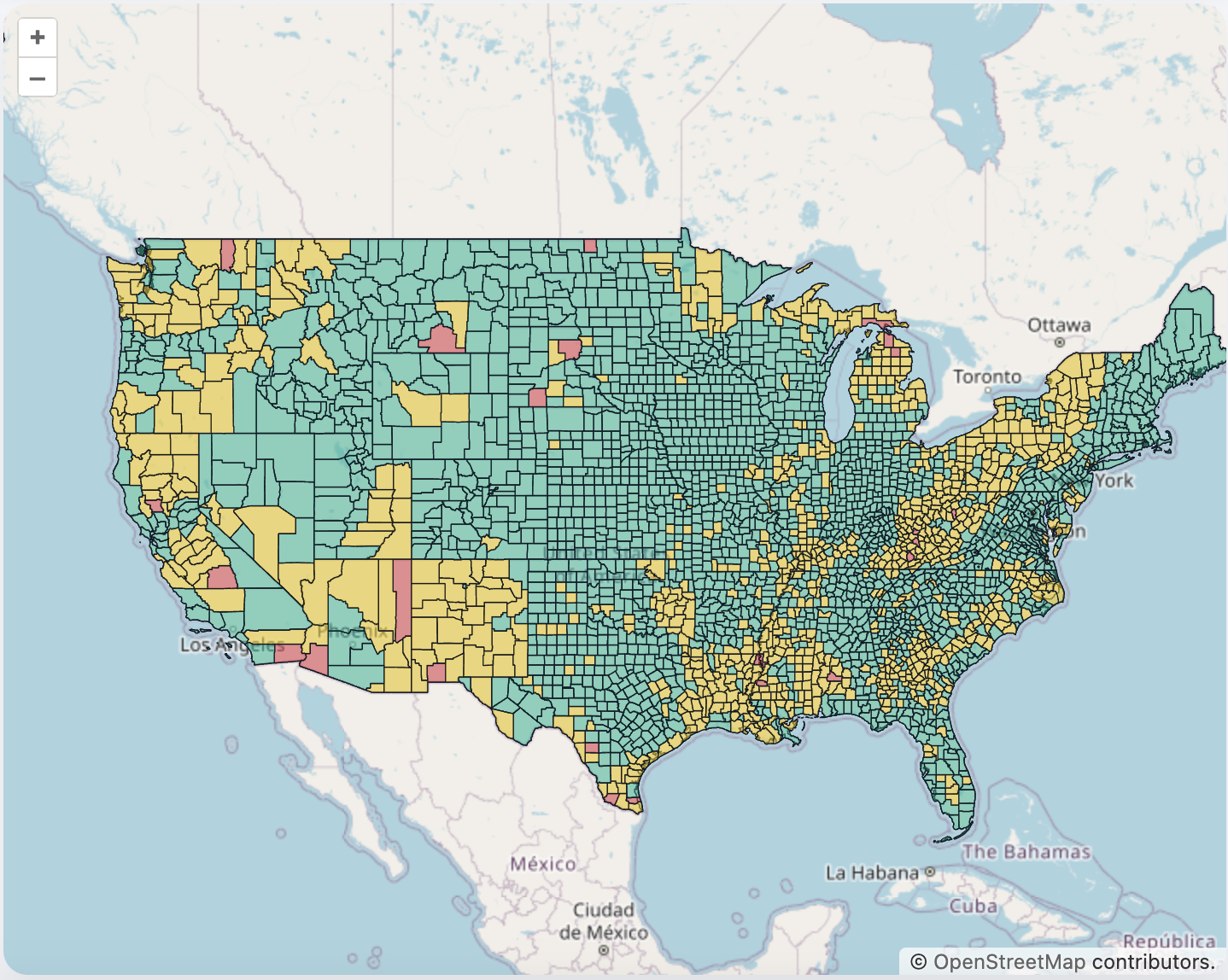
You see:

• When you pan/zoom to the data extent, features render above the base map.

Core snippet (script.js):

const layer1 = new ol.layer.Vector({ source: new ol.source.Vector({ url: 'gz\_2010\_us\_050\_00\_500k.json', format: new ol.format.GeoJSON() }) });

const layer2 = new ol.layer.Vector({ source: new ol.source.Vector({ url: 'data.geojson', format: new ol.format.GeoJSON() }) }); map.addLayer(layer1); map.addLayer(layer2);



**Step 5 — Wire Up Simple Filters (Checkbox → Layer Visibility)**

**Visibility toggles are the fastest UX win.**

You do:

• Add two checkboxes in the Filters panel and toggle layer visibility in JS.

You see:

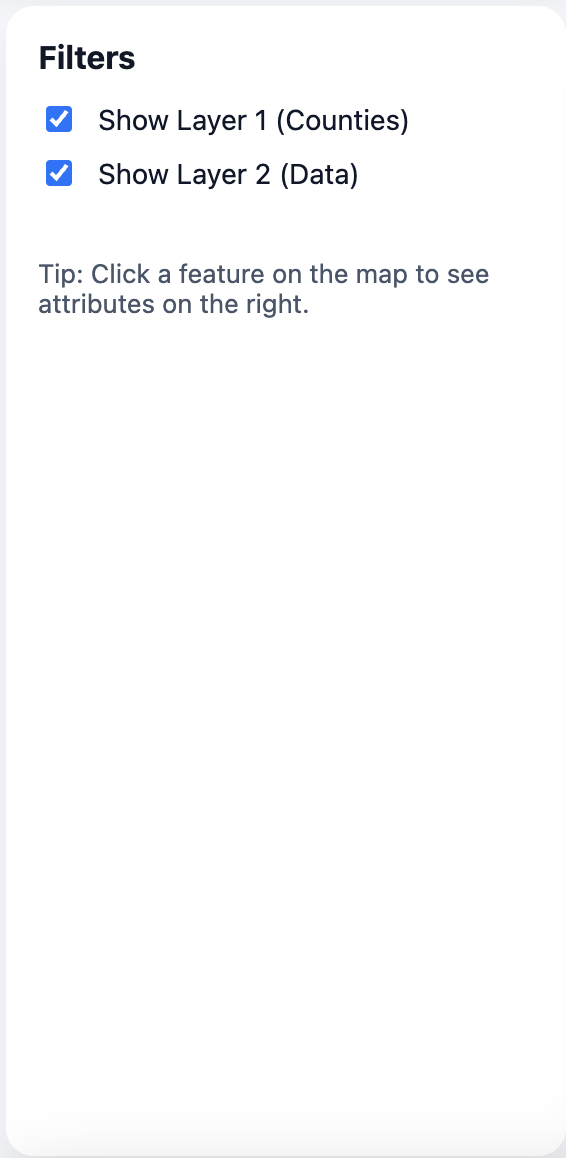
• Ticking/unticking instantly shows/hides each vector layer.

Core snippet (index.html Filters):

<label><input id="layer1" type="checkbox" checked> Show Layer 1</label> <label><input id="layer2" type="checkbox" checked> Show Layer 2</label>

Core snippet (script.js):

document.getElementById('layer1').onchange = e => layer1.setVisible(e.target.checked); document.getElementById('layer2').onchange = e => layer2.setVisible(e.target.checked);



**Step 6 — Inspector: Click → Lon/Lat + Attributes**

**Make your map interactively.**

You do:

• On single-click, read the clicked coordinate, find a feature under the cursor, and show its attributes.

You see:

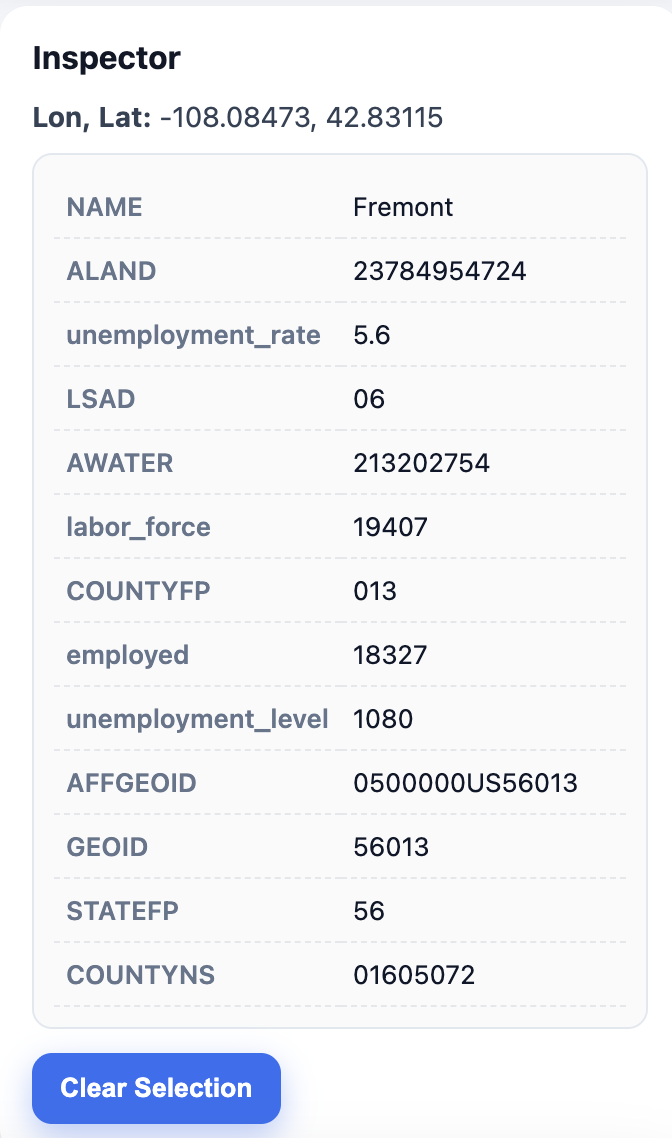
• The Inspector updates: Lon, Lat values plus a small list of key/value pairs from the feature.

Core snippet (script.js):

const lonlat = document.getElementById('lonlat');

const featureInfo = document.getElementById('feature-info');

map.on('singleclick', (evt) => { const f = map.forEachFeatureAtPixel(evt.pixel, (feat) => feat); const [lon, lat] = ol.proj.toLonLat(evt.coordinate); lonlat.textContent = `${lon.toFixed(5)}, ${lat.toFixed(5)}`; featureInfo.innerHTML = f ? Object.entries(f.getProperties()) .filter(([k]) => k !== 'geometry') .map(([k,v]) => `<div><b>${k}</b>: ${v}</div>`).join('') : '—'; });



**Step 7 — Optional Highlight (Nice-to-Have)**

**Make your UI better.**

You do:

• Add a lightweight highlight layer and clone the selected feature into it.

You see:

• A clear outline/fill around the clicked feature helps orientation.

Core snippet (script.js):

const highlight = new ol.layer.Vector({ source: new ol.source.Vector() }); map.addLayer(highlight); function showHighlight(f){ const src = highlight.getSource(); src.clear(); if (f) src.addFeature(f.clone()); } /\* call showHighlight(f) inside the singleclick handler after detecting f \*/

地图

描述已自动生成

**Step 8 — Run Locally and Sanity‑Check**

You do:

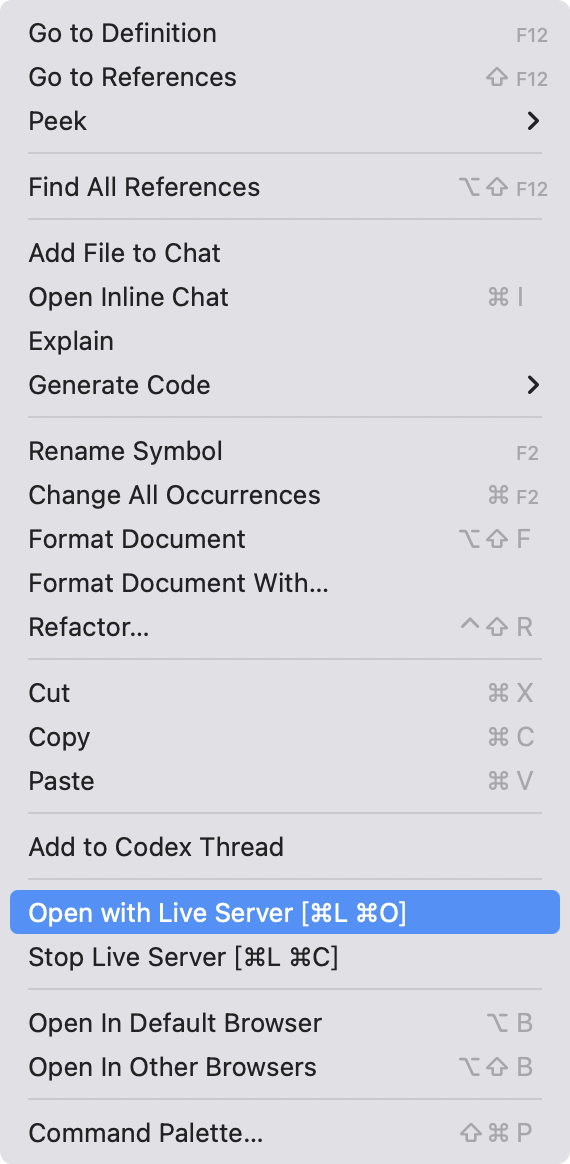
• Start a simple local server and open the site in your browser.

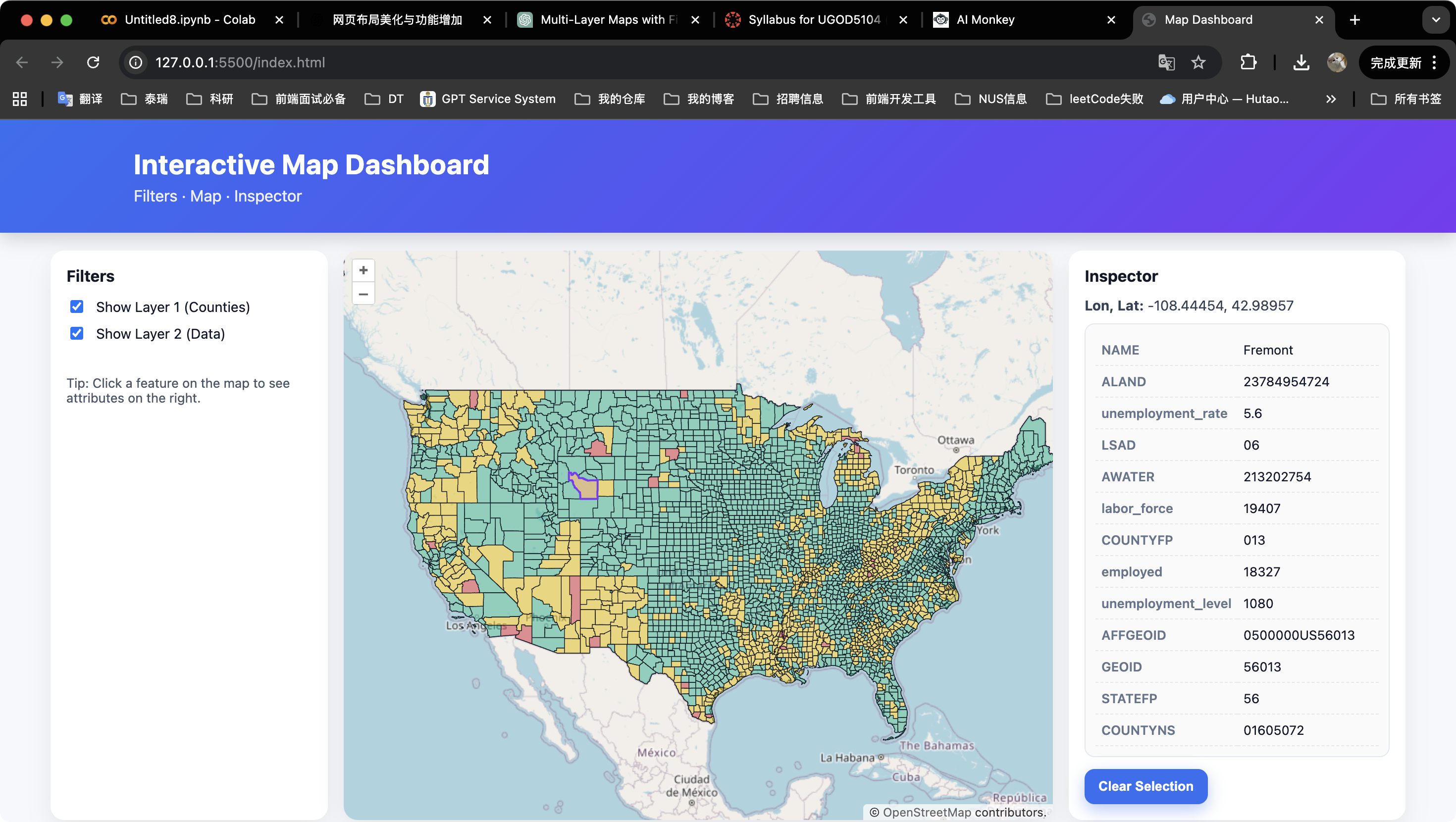
You see:

• Base map loads; layers toggle; click shows lon/lat and attributes; optional highlight draws.

Core step (terminal):

start your server via LIVE SERVER plugin in your VSCODE.





**Step 9 — Common Pitfalls and Fast Fixes**

• Blank map → Serve over HTTP, not file://, and check the console for 404s.

• No features → Verify GeoJSON paths; zoom to the data extent; confirm valid GeoJSON.

• Inspector empty → Click directly on a feature; ensure properties exist (non-null).

• Layers won’t toggle → Check element IDs match and that layer variables are in scope.

**What You’ve Learned**

• Structure first (HTML), behavior second (JS), polish last (CSS details).

• A minimal OL setup can scale: you can add legends, search, WMS/WFS, and exports without redesigning the core.