13.5.2

Map GeoJSON Point Type

You meet with Basil and Sadhana to discuss your project. Basil informs you that the earthquake data you'll map will have the geometry type Point. Basil thinks it would be a good idea to learn to parse GeoJSON data that is similar to the earthquake data.

Sadhana wants you to practice mapping GeoJSON data that she will give you to add to your `logic.js` file. This will be a good introduction on learning how to access the data from a JSON file.

Before we map any data, let's create a new branch called "Mapping_GeoJSON_Points" and create the following folder structure:

- Mapping_GeoJSON_Points
  - `index.html`
  - static
    - css
      - `style.css`
    - js
      - `config.js`
      - `logic.js`

Copy the necessary folders and files from one of your previous branches and add them to the Mapping_GeoJSON_Points folder.

Map a GeoJSON Point

First, we'll add single point on our map using GeoJSON data. The following GeoJSON data is a FeatureCollection object that has properties and geometry for the San Francisco Airport:

```
// Add GeoJSON data.
let sanFranAirport =
{"type":"FeatureCollection","features":[{
    "type":"Feature",
    "properties":{
        "id":"3469",
        "name":"San Francisco International Airport",
        "city":"San Francisco",
        "country":"United States",
        "faa":"SFO",
        "icao":"KSFO",
        "alt":"13",
        "tz-offset":"-8",
        "dst":"A",
        "tz":"America/Los_Angeles"},
        "geometry":{
            "type":"Point",
            "coordinates":[-122.375,37.61899948120117]}}
]};
```

Since we are going to add the San Francisco Airport to our map, let's change the center to the San Francisco Airport. Add the following code to our `logic.js` file to create the center of the map at the airport with a zoom level of "10."

```
// Create the map object with center at the San Francisco airport.
let map = L.map('mapid').setView([37.5, -122.5], 10);
```

In the **GeoJSON example** **(https://leafletjs.com/SlavaUkraini/examples/geojson/)** given on the Leaflet page, we can see that the simple GeoJSON feature is similar to our `sanFranAirport`.

```
var geojsonFeature = {
    "type": "Feature",
    "properties": {
        "name": "Coors Field",
        "amenity": "Baseball Stadium",
        "popupContent": "This is where the Rockies play!"
    },
    "geometry": {
        "type": "Point",
        "coordinates": [-104.99404, 39.75621]
    }
};
```

GeoJSON objects are added to the map through a GeoJSON layer, `L.geoJSON()`. In "The GeoJSON Layer" section, it says to create the GeoJSON layer and add it to our map. We can use the following code to do that:

```
L.geoJSON(geojsonFeature).addTo(map);
```

Let's edit this GeoJSON layer as follows:

```
// Grabbing our GeoJSON data.
L.geoJSON(sanFranAirport).addTo(map);
```

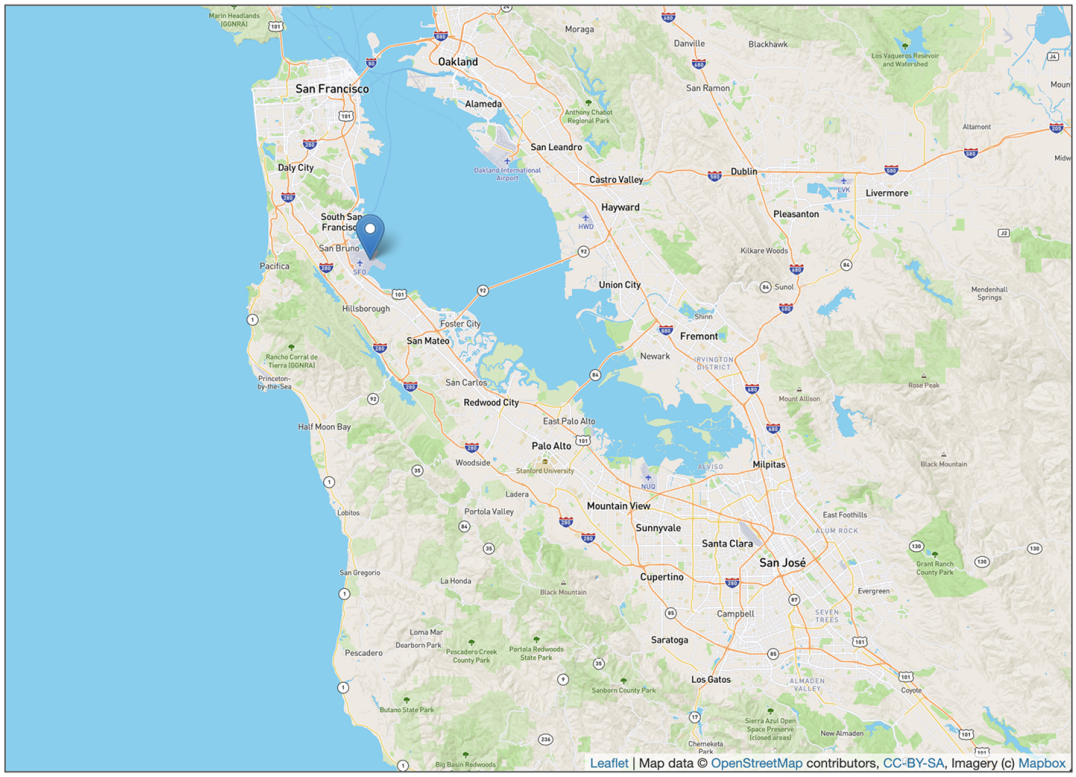
Also, add it to our `logic.js` file below the GeoJSON airport data and above the `tileLayer()` method. After you save the `logic.js` file, it should look like the following:

```
// Create the map object with center at the San Francisco airport.
let map = L.map('mapid').setView([37.5, -122.5], 10);

// Add GeoJSON data.
let sanFranAirport =
{"type":"FeatureCollection","features":[{
    "type":"Feature",
    "properties":{
        "id":"3469",
        "name":"San Francisco International Airport",
        "city":"San Francisco",
        "country":"United States",
        "faa":"SFO",
        "icao":"KSFO",
        "alt":"13",
        "tz-offset":"-8",
        "dst":"A",
        "tz":"America/Los_Angeles"},
        "geometry":{
            "type":"Point",
            "coordinates":[-122.375,37.61899948120117]}}
]};

// Grabbing our GeoJSON data.
L.geoJSON(sanFranAirport).addTo(map);
```

NOTE

Please note that the coordinates appear in reverse order `[-122.375, 37.61899948120117]`, compared to their order in the `setView()` method. This is because the GeoJSON data coordinates are set with the first parameter as X (longitude) and the second parameter as Y (latitude), as documented in the **GeoJSON Standard. (https://tools.ietf.org/html/rfc7946)** The `L.geoJSON()` layer reverses the coordinates to plot them on the map.

Open the `index.html` file in your browser. Your map should have a marker at SFO.



Later in this module we'll be using a URL to access a larger GeoJSON dataset to plot more points.

Bind a Popup to the Marker

REWIND

To display data on a map with a popup marker, we have to bind the marker with the GeoJSON layer, `L.geoJSON()`, using a callback function.

Our options to add data to a marker are to use the `pointToLayer` or `onEachFeature` callback functions. With either of these functions, we can add data to a map from each GeoJSON object. The major difference between the two functions is that the `pointToLayer` callback function adds markers to a map, whereas the `onEachFeature` callback function allows you to add styling and bind data to a popup marker.

Let's look at these two functions more closely.

The pointToLayer Function

For the `pointToLayer` callback function, the basic syntax for adding functionality to a marker is as follows:

```
L.geoJson(data, {
    pointToLayer: function(feature, latlng) {
      return L.marker(latlng);
     }
});
```

Let's break down what is happening in the `L.geoJSON()` layer:

1. We add two arguments: the `data` and the `pointToLayer` callback function.
2. The `data` will be our `sanFranAirport` data.
3. For the `pointToLayer` callback function, we are first going to call a `function()` where we pass each GeoJSON feature as `feature`, and its latitude and longitude as `latlng`.
4. Then we add a marker for each feature with a latitude and longitude in the `pointToLayer` callback function argument by using `return L.marker(latlng)`.

Retake

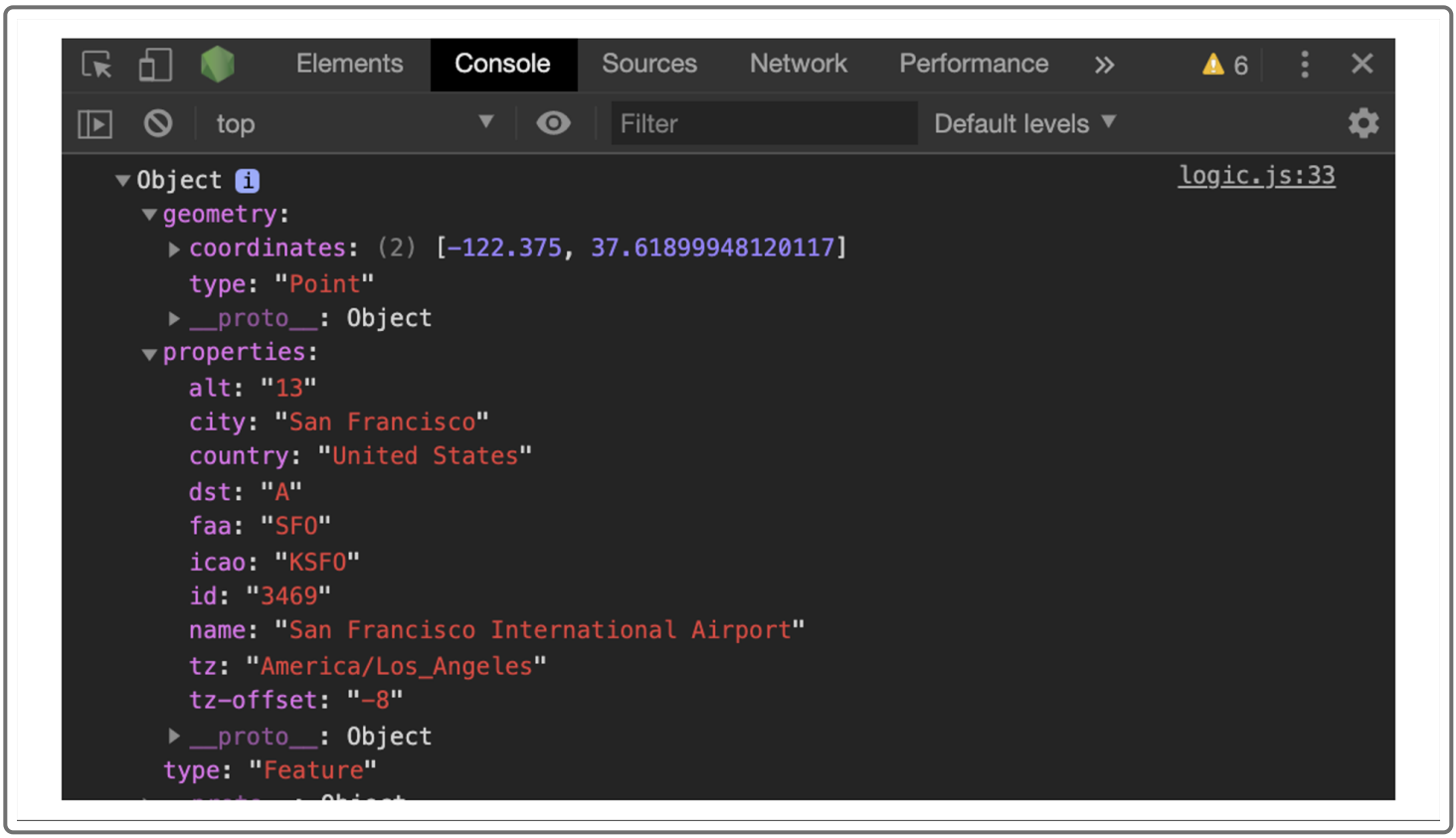
Even though we have a marker on the previous map, let's edit our `logic.js` file to add a marker using the `pointToLayer` function and add data to a popup marker.

First, let's edit the `logic.js` file to add the `pointToLayer` callback function to the `L.geoJSON()` layer. To better understand what is passed with the `feature` argument in the `function()`, we will add `feature` in the `console.log()` function. Edit your `L.geoJSON()` layer code to look like the following:

```
// Grabbing our GeoJSON data.
L.geoJSON(sanFranAirport, {
    // We turn each feature into a marker on the map.
    pointToLayer: function(feature, latlng) {
      console.log(feature);
      return L.marker(latlng);
    }
```

```
}).addTo(map);
```

Save your `logic.js` file and open the `index.html` file in your browser. The map should look the same as it did before the edits. However, if we open the console on our developer tools, we will see that the `feature` is the JavaScript object `geometry` and `properties` of our GeoJSON object.



Now, we'll add the data in the JavaScript objects to a popup marker.

REWIND

The properties in each JavaScript object can be accessed using the dot notation.

Retake

To add a popup marker, we need to use the `bindPopup()` method to the `pointToLayer` callback function. This will add a popup marker for each object in our GeoJSON data even though we only have one object in our data, SFO.

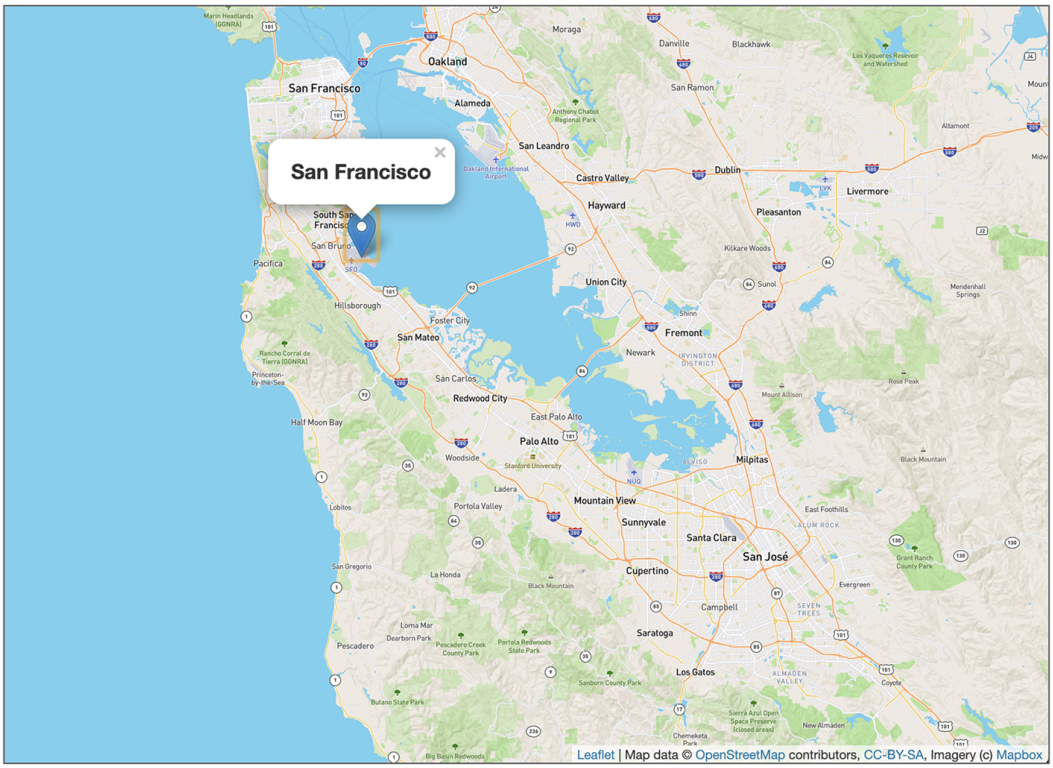
Let's add the city to the popup marker. In our `logic.js` file, after the `return L.marker(latlng)` in our `L.geoJSON()` layer, add the following code on the next line:

```
.bindPopup("<h2>" + feature.properties.city + "</h2>")
```

Using the dot notation, we can traverse through the JSON object to get the city by using `feature.properties.city`. Now, your `logic.js` file with `L.geoJSON()` layer should look like the following:

```
// Grabbing our GeoJSON data.
L.geoJson(sanFranAirport, {
    // We turn each feature into a Marker on the map.
    pointToLayer: function(feature, latlng) {
      console.log(feature);
      return L.marker(latlng)
      .bindPopup("<h2>" + feature.properties.city + "</h2>");
    }

  }).addTo(map);
```

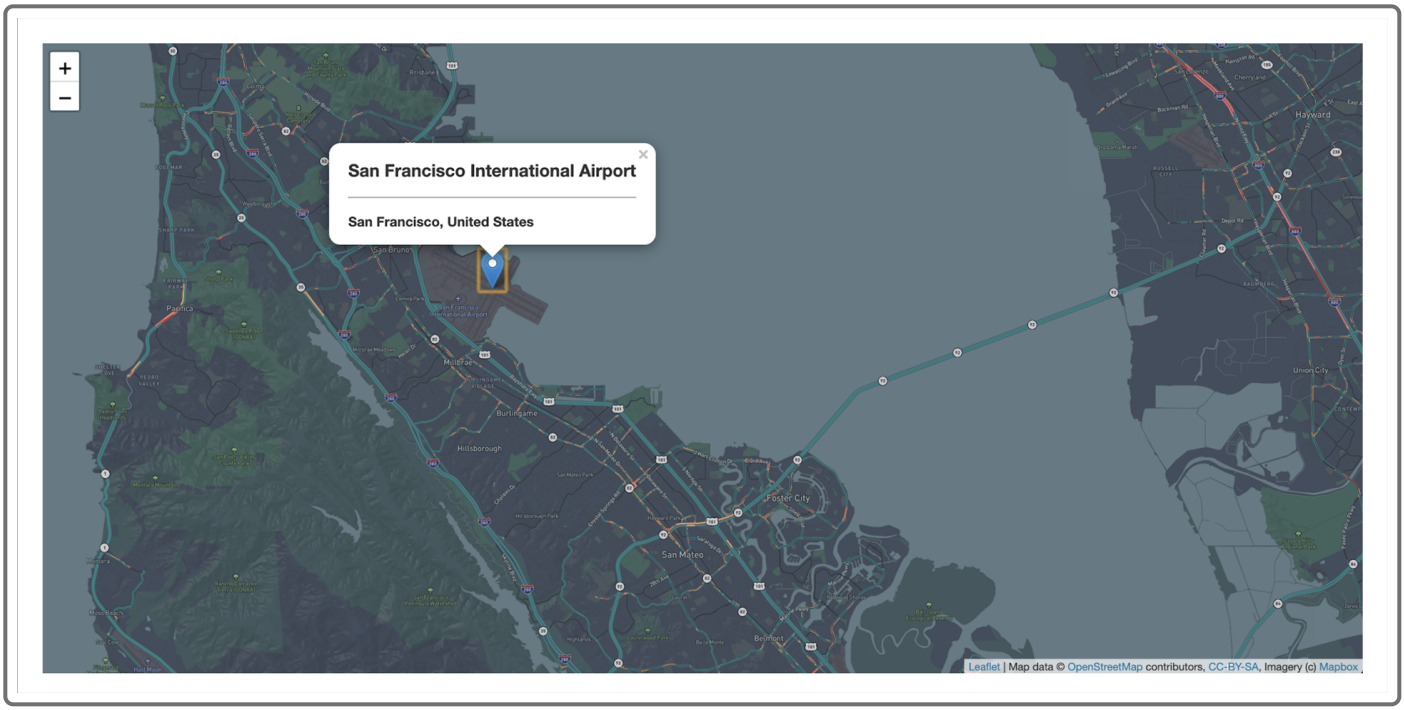
Our map should look like the following, where a marker, when clicked, shows a city name:



SKILL DRILL

Edit your `logic.js` to create a popup marker for San Francisco Airport on a night preview navigation map. When you click on the popup, it will display the city, state, and the name of the airport.

Your map should look like the following:



The onEachFeature Function

When we use the `onEachFeature` callback function we can add a popup marker for each feature and add data from the properties of the JavaScript object. The basic syntax for adding functionality to a marker is as follows:

```
L.geoJSON(data, {
    onEachFeature: function(feature, layer) {
      layer.bindPopup();
     }
});
```

Let's break down what is happening in the `L.geoJSON()` layer:

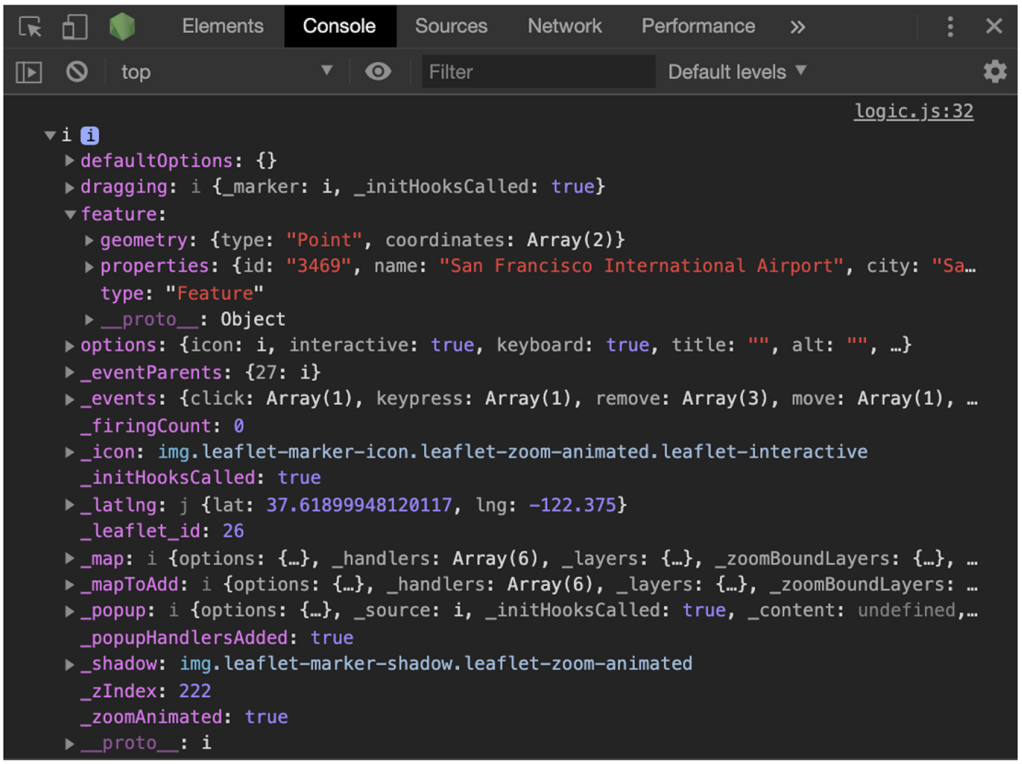
1. First, we add two arguments: the `data` and the `onEachFeature` callback function.
2. The `data` will be our `sanFranAirport` data.
3. With the `onEachFeature` callback function we are first going to call an anonymous function, `function()`, where we pass each GeoJSON feature as `feature`, and any properties to the second argument, `layer`.

Let's edit our `logic.js` file to add a popup marker using the `onEachFeature` function. First, edit the `logic.js` file to add the `onEachFeature` callback function to the `L.geoJSON()` layer. To see what is passed with the `layer` argument

in the anonymous `function()`, we'll pass `layer` in the `console.log()`function. Edit your `L.geoJSON()` layer code to look like the following:

```
// Grabbing our GeoJSON data.
L.geoJson(sanFranAirport, {
  onEachFeature: function(feature, layer) {
    console.log(layer);
    layer.bindPopup();
  }
}).addTo(map);
```

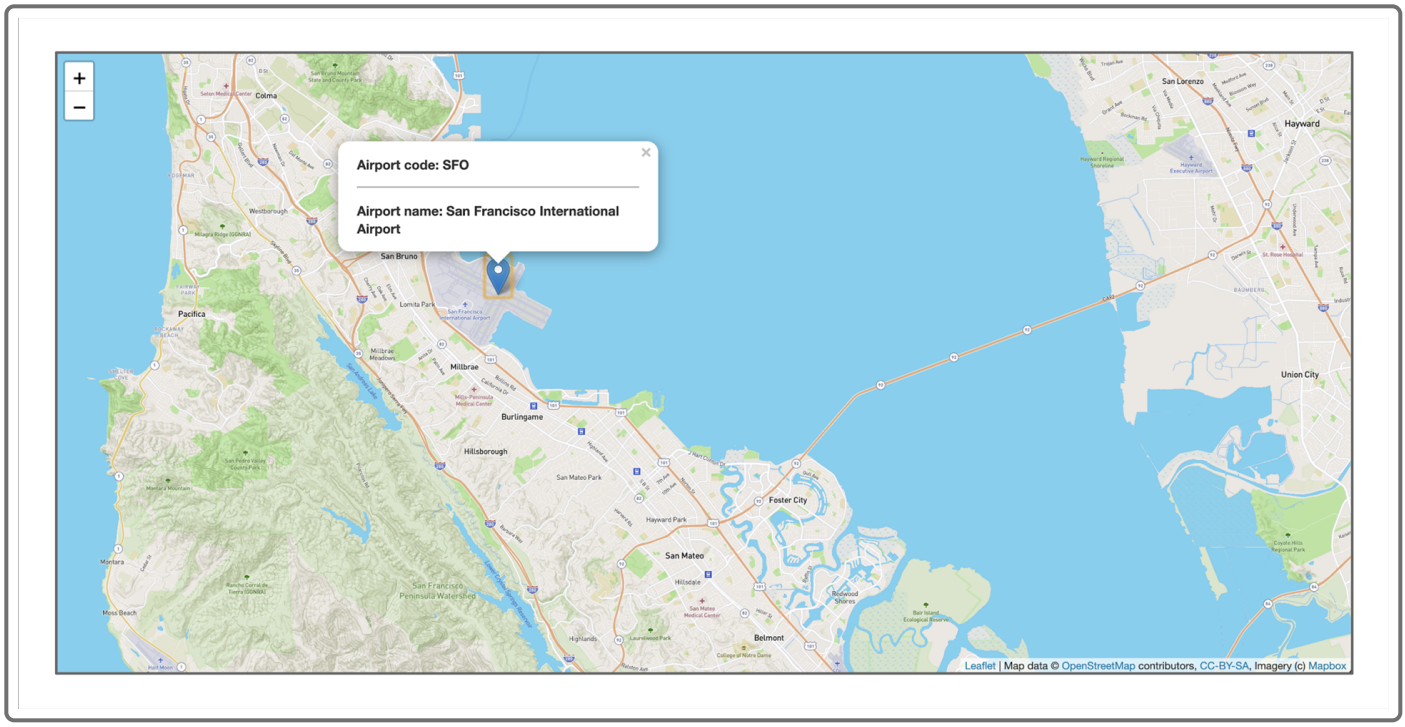
When we open our `index.html` file, the map will display a popup marker for SFO. When we open the console on our DevTools, we'll see that the `layer` returns many JavaScript methods that can be accessed and used, including the `geometry` and `properties` of our GeoJSON object.



SKILL DRILL

Edit your `logic.js` to create a popup marker for the San Francisco Airport on the outdoor map. When you click on the popup, it will display the airport code and name of the airport.

Your map should look like the following:



Great job on adding GeoJSON data to your map!

NOTE

For more information, see the **Leaflet documentation on the L.geoJSON() layer. (https://leafletjs.com/SlavaUkraini/reference-1.6.0.html#geojson)** .

Next, we'll map multiple point type geometry from a JSON file.