GeoMap Component Documentation

Alex-Ovidiu POPA

May 4, 2025

1 Introduction

The `GeoMap` component (Figure 1) is a React-based interactive map built using the Leaflet library and its React integration (`react-leaflet`). It provides functionalities such as marker management (adding, editing, deleting), marker clustering, and filtering by name. This document elaborates on each functionality and includes the corresponding code snippets.

2 Libraries Used

The following libraries are used in the `GeoMap` component:

- **Leaflet**: A JavaScript library for interactive maps. (`https://leafletjs.com/`)
- **react-leaflet**: React bindings for Leaflet. (`https://react-leaflet.js.org/`)
- **react-leaflet-markercluster**: A plugin for clustering markers on the map. (`https://github.com/yuzhva/react-leaflet-markercluster`)
- **@changey/react-leaflet-markercluster**: A modern fork of the marker cluster plugin.

3 Main Functionalities

The `GeoMap` component provides the following features:

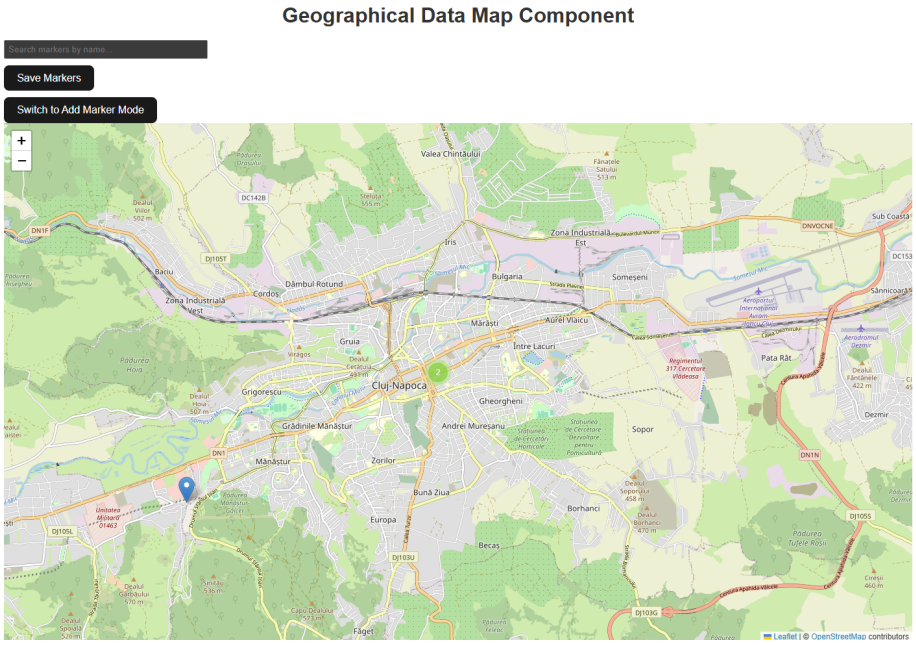

Figure 1: Embedded view of the GeoMap component

3.1 Marker Management

Description: Users can add, edit, and delete markers on the map. Markers are stored in the component's state and can be persisted using `localStorage`.

Add Marker: Markers can be added by clicking on the map in "Add Marker Mode." The following function handles adding a new marker:

```
const addMarker = () => {
if (newMarkerPosition && newMarkerName.trim()) {
    setMarkers([...markers, { position: newMarkerPosition
        , name: newMarkerName }]);
    setNewMarkerPosition(null);
    setNewMarkerName('');
}
};
```

Edit Marker: Markers can be edited by selecting them and modifying their name or position:

```
const saveEditedMarker = () => {
if (editingMarker !== null && editedPosition) {
    const updatedMarkers = [...markers];
```

```
    updatedMarkers[editingMarker] = {
        name: editedName,
        position: editedPosition,
    };
    setMarkers(updatedMarkers);
    setEditingMarker(null);
    setEditedName('');
    setEditedPosition(null);
}
};
```

Delete Marker: Markers can be removed from the map using the following function:

```
const deleteMarker = (index: number) => {
const updatedMarkers = markers.filter((_, i) => i !==
   index);
setMarkers(updatedMarkers);
};
```

3.2 Marker Clustering

Description: Markers are automatically grouped into clusters for better visualization when there are many markers on the map. This is achieved using the `react-leaflet-markercluster` library.

Implementation:

```
<MarkerClusterGroup>
{filteredMarkers.map((marker, index) => (
    <Marker key={index} position={marker.position}>
        <Popup>
            <p>{marker.name}</p>
        </Popup>
    </Marker>
))}
</MarkerClusterGroup>
```

3.3 Search Functionality

Description: Users can filter markers by name using a search input. The search query is matched against the marker names.

Implementation:

```
const filteredMarkers = markers.filter((marker) =>
marker.name.toLowerCase().includes(searchQuery.
    toLowerCase())
);
```

3.4 Mode Switching

Description: The map can toggle between "Add Marker Mode" (for adding markers) and "Move Mode" (for navigating the map). In "Add Marker Mode," dragging is disabled.

Implementation:

```
const MapModeHandler: React.FC<{ isAddMarkerMode: boolean
    }> = ({ isAddMarkerMode }) => {
const map = useMap();

React.useEffect(() => {
    if (isAddMarkerMode) {
        map.dragging.disable();
    } else {
        map.dragging.enable();
    }
}, [isAddMarkerMode, map]);

return null;
};
```

3.5 Persistent Storage

Description: Markers can be saved to and loaded from `localStorage`, ensuring persistence across sessions. Although a trivial approach, this may be replaced in real-world applications with an API call, if desired.

Save Markers:

```
const saveMarkers = () => {
localStorage.setItem('markers', JSON.stringify(markers));
alert('Markers␣saved␣successfully!');
};
```

Load Markers:

```
useEffect(() => {
const savedMarkers = localStorage.getItem('markers');
```

```
if (savedMarkers) {
    setMarkers(JSON.parse(savedMarkers));
}
}, []);
```

4 Component Integration

In order to integrate the component within a React application, one must simply import it, and use it as shown below:

```
import GeoMap from './components/GeoMap';

const App: React.FC = () => {
    return (
        <div>
            <h1>Geographical Data Map Component</h1>
            <GeoMap />
        </div>
    );
};
```