Comprehensive Guide to Leaflet.js with React

Introduction

Welcome to this comprehensive guide on integrating Leaflet.js with React! Whether you're building a location-based service, visualizing geographical data, or simply adding an interactive map to your application, this guide will walk you through everything you need to know.

Leaflet.js is a powerful, lightweight JavaScript library for interactive maps. When combined with React's component-based architecture, you can create maintainable, efficient map interfaces that respond dynamically to user interactions and data changes.

This guide assumes you're already familiar with React concepts but have no prior experience with Leaflet or mapping technologies. We'll build your knowledge progressively, starting with fundamentals and moving to advanced features.

Table of Contents

Leaflet Fundamentals

What is Leaflet?

Leaflet is an open-source JavaScript library for mobile-friendly interactive maps. Created by Vladimir Agafonkin in 2011, it's become the leading mapping library for several key reasons:

- **Lightweight**: The core library is only around 39KB of JS, making it much smaller than alternatives like Google Maps or OpenLayers.
- **Simplicity**: It has a clean, readable API that makes it easy to learn and use.
- **Performance**: It's designed to work efficiently even on older mobile devices.
- **Extensibility**: A vibrant plugin ecosystem allows for adding specialized functionality.
- **Open Source**: No API keys or usage limits to worry about (though some tile providers do require keys).

Leaflet follows a modular design that makes it perfect for integration with React's component-based architecture. Each map element (markers, popups, polygons) is an object that can be individually controlled and styled.

Core Concepts

Before diving into code, let's understand the fundamental concepts in web mapping:

Maps and Coordinate Systems

At its core, Leaflet creates an interactive map that users can pan and zoom. Maps in Leaflet use the Web Mercator projection (EPSG:3857), which is the standard for web maps. Coordinates are expressed as [latitude,

longitude] pairs, where:

- **Latitude**: Ranges from -90° (South Pole) to 90° (North Pole)
- **Longitude**: Ranges from -180° (West) to 180° (East)

Leaflet handles the conversion between geographical coordinates and pixel coordinates on the screen.

Tile Layers

Web maps are typically composed of "tiles" - small square images (usually 256x256 pixels) that are stitched together to form the complete map. As you zoom in or out, different sets of tiles are loaded at different resolutions.

These tiles can come from various providers:

- OpenStreetMap (free, community-maintained)
- Mapbox (commercial, highly customizable)
- Google Maps (commercial)
- And many others

Markers and Popups

- **Markers**: Icons placed at specific coordinates on the map, typically used to indicate points of interest.
- **Popups**: Information bubbles that appear when elements on the map are clicked.

Layers

Everything visible on a Leaflet map belongs to a layer:

- **Base layers**: The background map (usually tile layers)
- **Overlay layers**: Additional information displayed on top of the base layer (markers, lines, polygons, etc.)

Layers can be added, removed, and toggled, allowing for dynamic map behavior.

Setting up Leaflet in React

To use Leaflet with React, we'll leverage the `react-leaflet` library, which provides React components that wrap Leaflet elements, making them easier to integrate into React applications.

Installation

First, install the necessary packages:

```
npm install leaflet react-leaflet
# or with yarn
yarn add leaflet react-leaflet
```

You'll also need to include Leaflet's CSS. Add this to your application's entry point (e.g., `index.js` or `App.js`):

```
import 'leaflet/dist/leaflet.css';
```

Fixing the Marker Icon Issue

One common issue when setting up Leaflet in React is missing marker icons. This happens because the default marker icons in Leaflet reference images using relative paths, which don't work properly with React's build system.

Here's how to fix it:

```
// At the top of your component file or in a separate utilities file
import L from 'leaflet';
import icon from 'leaflet/dist/images/marker-icon.png';
import iconShadow from 'leaflet/dist/images/marker-shadow.png';

let DefaultIcon = L.icon({
  iconUrl: icon,
  shadowUrl: iconShadow,
  iconSize: [25, 41],
  iconAnchor: [12, 41],
  popupAnchor: [1, -34],
  shadowSize: [41, 41],
});

L.Marker.prototype.options.icon = DefaultIcon;
```

Your First Leaflet Map in React

Now, let's create a basic map component:

```
import React from 'react';
import { MapContainer, TileLayer, Marker, Popup } from 'react-leaflet';
import 'leaflet/dist/leaflet.css';

// Import this if you're using the icon fix from above
import './leaflet-icon-fix'; // Adjust the path as needed

const MapComponent = () => {
  // Set the center coordinates for the map
  const position = [51.505, -0.09]; // London

  return (
    <div className="map-container" style={{ height: '500px', width: '100%' }}>
      <MapContainer
        center={position}
        zoom={13}
        style={{ height: '100%', width: '100%' }}
```

```
      >
        {/* Base tile layer - this is the actual map */}
        <TileLayer
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        />

        {/* A simple marker with a popup */}
        <Marker position={position}>
          <Popup>
            A pretty CSS3 popup. <br /> Easily customizable.
          </Popup>
        </Marker>
      </MapContainer>
    </div>
  );
};

export default MapComponent;
```

In this example:

- `MapContainer` is the main component that creates the map.
- `TileLayer` defines the map's visual appearance.
- `Marker` places a pin at the specified coordinates.
- `Popup` creates a clickable information bubble attached to the marker.

Component Lifecycle Considerations

Using Leaflet with React requires some special considerations due to how these libraries handle the DOM.

The ref Pattern

Leaflet directly manipulates the DOM, while React uses a virtual DOM. To use Leaflet's imperative methods, you'll often need to access the underlying Leaflet objects using refs.

```
import React, { useEffect, useRef } from 'react';
import { MapContainer, TileLayer, useMap } from 'react-leaflet';

// This is a separate component that can access the map instance
const MapController = ({ center, zoom }) => {
  const map = useMap();

  useEffect(() => {
    map.setView(center, zoom);
  }, [center, zoom, map]);

  return null;
};
```

```
const MapWithControl = ({ center, zoom }) => {
  return (
    <div style={{ height: '500px', width: '100%' }}>
      <MapContainer
        center={center}
        zoom={zoom}
        style={{ height: '100%', width: '100%' }}
      >
        <TileLayer
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        />
        <MapController center={center} zoom={zoom} />
      </MapContainer>
    </div>
  );
};

export default MapWithControl;
```

Cleanup to Prevent Memory Leaks

Always clean up Leaflet objects when components unmount:

```
import React, { useEffect } from 'react';
import L from 'leaflet';
import { useMap } from 'react-leaflet';

const LeafletPlugin = () => {
  const map = useMap();

  useEffect(() => {
    // Initialize some Leaflet plugin or object
    const someControl = L.control.layers().addTo(map);

    // Cleanup function to remove it when component unmounts
    return () => {
      someControl.remove();
    };
  }, [map]);

  return null;
};
```

Handling Events

React-Leaflet provides event handlers for common Leaflet events:

```
import React from 'react';
import { MapContainer, TileLayer, useMapEvents } from 'react-leaflet';

const EventHandler = () => {
  const map = useMapEvents({
    click: (e) => {
      console.log('Map clicked at:', e.latlng);
    },
    zoom: () => {
      console.log('Zoom level:', map.getZoom());
    },
  });

  return null;
};

const InteractiveMap = () => {
  return (
    <MapContainer
      center={[51.505, -0.09]}
      zoom={13}
      style={{ height: '500px' }}
    >
      <TileLayer
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
      />
      <EventHandler />
    </MapContainer>
  );
};

export default InteractiveMap;
```

Basic Map Configuration

Now let's explore the key configuration options for your map:

Essential MapContainer Properties

```
<MapContainer
  center={[51.505, -0.09]} // Initial center position [lat, lng]
  zoom={13} // Initial zoom level (0-18, where 18 is closest)
  scrollWheelZoom={true} // Enable/disable zoom via mouse wheel
  doubleClickZoom={true} // Enable/disable zoom via double click
  dragging={true} // Enable/disable map dragging
  zoomControl={true} // Show/hide zoom control buttons
  attributionControl={true} // Show/hide attribution
  style={{ height: '500px', width: '100%' }} // Container styling
```

```
>
  {/* Map content goes here */}
</MapContainer>
```

Setting Map Boundaries

You can restrict the areas users can pan to and the zoom levels they can use:

```
import { useMap } from 'react-leaflet';
import { useEffect } from 'react';

const MapBoundaries = () => {
  const map = useMap();

  useEffect(() => {
    // Restrict map panning to these coordinates
    const southWest = L.latLng(40.712, -74.227);
    const northEast = L.latLng(40.774, -74.125);
    const bounds = L.latLngBounds(southWest, northEast);

    map.setMaxBounds(bounds);
    map.setMinZoom(12);
    map.setMaxZoom(16);

    // Makes the map "bounce back" when user pans outside bounds
    map.on('drag', () => {
      map.panInsideBounds(bounds, { animate: false });
    });
  }, [map]);

  return null;
};
```

Multiple Map Views

If you need to switch between different map views, manage the configuration in state:

```
import React, { useState } from 'react';
import { MapContainer, TileLayer, useMap } from 'react-leaflet';

// Component to update the map view
const ChangeView = ({ center, zoom }) => {
  const map = useMap();
  map.setView(center, zoom);
  return null;
};

const MapWithLocations = () => {
  const locations = {
```

```
    london: { center: [51.505, -0.09], zoom: 13 },
    paris: { center: [48.8566, 2.3522], zoom: 12 },
    tokyo: { center: [35.6762, 139.6503], zoom: 11 },
  };

  const [currentView, setCurrentView] = useState('london');

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <button onClick={() => setCurrentView('london')}>London</button>
        <button onClick={() => setCurrentView('paris')}>Paris</button>
        <button onClick={() => setCurrentView('tokyo')}>Tokyo</button>
      </div>

      <MapContainer
        center={locations[currentView].center}
        zoom={locations[currentView].zoom}
        style={{ height: '500px', width: '100%' }}
      >
        <TileLayer
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        />
        <ChangeView
          center={locations[currentView].center}
          zoom={locations[currentView].zoom}
        />
      </MapContainer>
    </div>
  );
};

export default MapWithLocations;
```

Common Pitfalls and Solutions

1. **Map not displaying**
   - Ensure the container has a defined height (Leaflet can't calculate dimensions from percentage-based heights without a parent height)
   - Check that Leaflet CSS is properly imported
2. **Markers not appearing**
   - Make sure you've fixed the icon path issue mentioned earlier
3. **Map rerenders causing performance issues**
   - Move state updates and event handlers to child components where possible
   - Use memoization for expensive calculations

4. **MapContainer not updating with new props**
    - `MapContainer` doesn't update dynamically from props; use the `useMap` hook in a child component as shown above

Map Layers and Controls

Types of Layers

Leaflet supports various types of layers that serve different purposes. Understanding these layer types is crucial for creating effective maps.

Tile Layers

Tile layers form the base map and come in various styles:

```
import React, { useState } from 'react';
import { MapContainer, TileLayer } from 'react-leaflet';

const TileLayerExample = () => {
  // Different tile layer options
  const tileLayers = {
    openStreetMap: {
      url: 'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
      attribution:
        '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
    },
    openTopoMap: {
      url: 'https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png',
      attribution:
        'Map data: &copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors, <a
href="http://viewfinderpanoramas.org">SRTM</a> | Map style: &copy; <a
href="https://opentopomap.org">OpenTopoMap</a>',
    },
    stamenTerrain: {
      url: 'https://stamen-tiles-{s}.a.ssl.fastly.net/terrain/{z}/{x}/{y}{r}.png',
      attribution:
        'Map tiles by <a href="http://stamen.com">Stamen Design</a>, <a
href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a> &mdash; Map data
&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors',
    },
  };

  const [currentLayer, setCurrentLayer] = useState('openStreetMap');

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <button onClick={() => setCurrentLayer('openStreetMap')}>
```

```
            OpenStreetMap
          </button>
          <button onClick={() => setCurrentLayer('openTopoMap')}>
            Topographic
          </button>
          <button onClick={() => setCurrentLayer('stamenTerrain')}>
            Terrain
          </button>
        </div>

        <MapContainer
          center={[51.505, -0.09]}
          zoom={13}
          style={{ height: '500px', width: '100%' }}
        >
          <TileLayer
            url={tileLayers[currentLayer].url}
            attribution={tileLayers[currentLayer].attribution}
          />
        </MapContainer>
      </div>
    );
};

export default TileLayerExample;
```

WMS Layers

Web Map Service (WMS) layers are different from standard tile layers. They're dynamic map images generated by a server based on geographical data:

```
import React from 'react';
import { MapContainer, WMSTileLayer, TileLayer } from 'react-leaflet';

const WMSLayerExample = () => {
  return (
    <MapContainer
      center={[51.505, -0.09]}
      zoom={13}
      style={{ height: '500px', width: '100%' }}
    >
      {/* Base map */}
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
      />

      {/* WMS overlay */}
      <WMSTileLayer
        url="http://mesonet.agron.iastate.edu/cgi-bin/wms/nexrad/n0r.cgi"
```

```
        layers="nexrad-n0r-900913"
        format="image/png"
        transparent={true}
        attribution="Weather data © 2012 IEM Nexrad"
      />
    </MapContainer>
  );
};

export default WMSLayerExample;
```

Image Overlays

Image overlays allow you to display an image at specific geographical bounds:

```
import React from 'react';
import { MapContainer, TileLayer, ImageOverlay } from 'react-leaflet';
import L from 'leaflet';

const ImageOverlayExample = () => {
  // Image URL
  const imageUrl =
    'https://maps.lib.utexas.edu/maps/historical/newark_nj_1922.jpg';

  // Image bounds (coordinates where the image should be placed)
  const bounds = L.latLngBounds([40.712216, -74.22655], [40.773941, -74.12544]);

  return (
    <MapContainer
      center={[40.74, -74.18]}
      zoom={13}
      style={{ height: '500px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
      />

      <ImageOverlay url={imageUrl} bounds={bounds} opacity={0.8} zIndex={10} />
    </MapContainer>
  );
};

export default ImageOverlayExample;
```

Vector Layers

Vector layers include markers, lines, polygons, and other geometric shapes. We'll cover these in more detail in the Geometry and Vector Features section.

Dynamic Layer Management

Managing layers dynamically is a common requirement in interactive maps. Let's see how to add, remove, and update layers based on user actions or data changes.

Adding and Removing Layers

```
import React, { useState } from 'react';
import { MapContainer, TileLayer, Marker, Popup } from 'react-leaflet';

const DynamicLayersExample = () => {
  // Sample data for markers
  const initialMarkers = [
    { id: 1, position: [51.505, -0.09], name: 'London' },
    { id: 2, position: [48.8566, 2.3522], name: 'Paris' },
    { id: 3, position: [41.9028, 12.4964], name: 'Rome' },
  ];

  const [markers, setMarkers] = useState(initialMarkers);
  const [newMarkerName, setNewMarkerName] = useState('');

  // Add a new marker at a random position near London
  const addMarker = () => {
    if (!newMarkerName) return;

    // Generate random position near London
    const randomLat = 51.505 + (Math.random() - 0.5) * 0.1;
    const randomLng = -0.09 + (Math.random() - 0.5) * 0.1;

    const newMarker = {
      id: Date.now(), // Use timestamp as unique ID
      position: [randomLat, randomLng],
      name: newMarkerName,
    };

    setMarkers([...markers, newMarker]);
    setNewMarkerName('');
  };

  // Remove a marker by ID
  const removeMarker = (id) => {
    setMarkers(markers.filter((marker) => marker.id !== id));
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <input
          type="text"
```

```
            value={newMarkerName}
            onChange={(e) => setNewMarkerName(e.target.value)}
            placeholder="Enter marker name"
          />
          <button onClick={addMarker}>Add Marker</button>
        </div>

        <MapContainer
          center={[51.505, -0.09]}
          zoom={13}
          style={{ height: '500px', width: '100%' }}
        >
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />

          {/* Render markers dynamically from state */}
          {markers.map((marker) => (
            <Marker key={marker.id} position={marker.position}>
              <Popup>
                <div>
                  <h3>{marker.name}</h3>
                  <button onClick={() => removeMarker(marker.id)}>Remove</button>
                </div>
              </Popup>
            </Marker>
          ))}
        </MapContainer>

        <div style={{ marginTop: '10px' }}>
          <h3>Current Markers:</h3>
          <ul>
            {markers.map((marker) => (
              <li key={marker.id}>
                {marker.name} ({marker.position[0].toFixed(4)},{' '}
                {marker.position[1].toFixed(4)})
                <button onClick={() => removeMarker(marker.id)}>Remove</button>
              </li>
            ))}
          </ul>
        </div>
      </div>
    );
};

export default DynamicLayersExample;
```

Updating Layer Properties

Sometimes you need to update properties of existing layers without completely removing and re-adding them:

```
import React, { useState, useEffect } from 'react';
import { MapContainer, TileLayer, Circle, Popup } from 'react-leaflet';

const UpdateLayerPropertiesExample = () => {
  const [circles, setCircles] = useState([
    { id: 1, center: [51.505, -0.09], radius: 500, color: '#ff0000' },
    { id: 2, center: [51.51, -0.1], radius: 300, color: '#0000ff' },
  ]);

  // Effect to randomly update circle properties every 2 seconds
  useEffect(() => {
    const interval = setInterval(() => {
      setCircles((prevCircles) =>
        prevCircles.map((circle) => ({
          ...circle,
          // Randomly adjust the radius
          radius: Math.max(100, circle.radius + (Math.random() - 0.5) * 100),
          // Randomly change the color
          color: `#${Math.floor(Math.random() * 16777215).toString(16)}`,
        }))
      );
    }, 2000);

    return () => clearInterval(interval);
  }, []);

  return (
    <MapContainer
      center={[51.505, -0.09]}
      zoom={13}
      style={{ height: '500px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
      />

      {circles.map((circle) => (
        <Circle
          key={circle.id}
          center={circle.center}
          radius={circle.radius}
          pathOptions={{ fillColor: circle.color, color: circle.color }}
        >
          <Popup>
            Circle {circle.id}
            <br />
            Radius: {circle.radius}m<br />
```

```
              Color: {circle.color}
            </Popup>
          </Circle>
        ))}
      </MapContainer>
    );
};

export default UpdateLayerPropertiesExample;
```

Layer Groups and Control

Layer groups allow you to manage multiple layers as a single unit. Layer controls provide UI elements for users to toggle layers.

Layer Groups

```
import React from 'react';
import {
  MapContainer,
  TileLayer,
  LayerGroup,
  Circle,
  FeatureGroup,
  Rectangle,
} from 'react-leaflet';
import L from 'leaflet';

const LayerGroupExample = () => {
  // Define bounds for the rectangle
  const rectangle = [
    [51.49, -0.08],
    [51.5, -0.06],
  ];

  return (
    <MapContainer
      center={[51.505, -0.09]}
      zoom={13}
      style={{ height: '500px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
      />

      {/* Simple LayerGroup example */}
      <LayerGroup>
        <Circle
          center={[51.505, -0.09]}
```

```
            radius={200}
            pathOptions={{ color: 'red' }}
          />
          <Circle
            center={[51.51, -0.1]}
            radius={100}
            pathOptions={{ color: 'red' }}
          />
        </LayerGroup>

        {/* FeatureGroup has additional functionality like bindPopup, bindTooltip
*/}
        <FeatureGroup pathOptions={{ color: 'purple' }}>
          <Popup>Popup for feature group</Popup>
          <Circle center={[51.51, -0.08]} radius={200} />
          <Rectangle bounds={rectangle} />
        </FeatureGroup>
      </MapContainer>
    );
};

export default LayerGroupExample;
```

Layer Control

```
import React from 'react';
import {
  MapContainer,
  TileLayer,
  LayersControl,
  Marker,
  Popup,
  Circle,
  FeatureGroup,
} from 'react-leaflet';

const { BaseLayer, Overlay } = LayersControl;

const LayerControlExample = () => {
  return (
    <MapContainer
      center={[51.505, -0.09]}
      zoom={13}
      style={{ height: '500px', width: '100%' }}
    >
      <LayersControl position="topright">
        {/* Base layers (only one can be active at a time) */}
        <BaseLayer checked name="OpenStreetMap">
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
```

```
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />
        </BaseLayer>

        <BaseLayer name="OpenTopoMap">
          <TileLayer
            url="https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png"
            attribution='Map data: &copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors, <a
href="http://viewfinderpanoramas.org">SRTM</a> | Map style: &copy; <a
href="https://opentopomap.org">OpenTopoMap</a>'
          />
        </BaseLayer>

        {/* Overlay layers (can be toggled on/off independently) */}
        <Overlay name="Important Places">
          <LayerGroup>
            <Marker position={[51.505, -0.09]}>
              <Popup>London City Center</Popup>
            </Marker>
            <Marker position={[51.51, -0.1]}>
              <Popup>Regent's Park</Popup>
            </Marker>
          </LayerGroup>
        </Overlay>

        <Overlay name="Points of Interest">
          <FeatureGroup>
            <Circle
              center={[51.508, -0.11]}
              radius={500}
              pathOptions={{ color: 'red' }}
            />
            <Circle
              center={[51.499, -0.08]}
              radius={300}
              pathOptions={{ color: 'blue' }}
            />
          </FeatureGroup>
        </Overlay>
      </LayersControl>
    </MapContainer>
  );
};

export default LayerControlExample;
```

Custom Controls

You can create custom controls to add your own UI elements to the map.

Basic Custom Control

```
import React, { useEffect } from 'react';
import { MapContainer, TileLayer, useMap } from 'react-leaflet';
import L from 'leaflet';
import './CustomControl.css'; // You'll need to create this CSS file

// Component to add a custom control to the map
const ZoomInfo = () => {
  const map = useMap();

  useEffect(() => {
    // Create a custom control
    const customControl = L.Control.extend({
      options: {
        position: 'bottomleft',
      },

      onAdd: function () {
        const container = L.DomUtil.create('div', 'custom-control');
        container.innerHTML = `Current zoom: ${map.getZoom()}`;

        // Update the content when zoom changes
        map.on('zoomend', () => {
          container.innerHTML = `Current zoom: ${map.getZoom()}`;
        });

        return container;
      },
    });

    // Add the control to the map
    const zoomInfoControl = new customControl();
    map.addControl(zoomInfoControl);

    // Clean up on unmount
    return () => {
      map.removeControl(zoomInfoControl);
    };
  }, [map]);

  return null;
};

// CSS for custom control (CustomControl.css)
/*
.custom-control {
  padding: 6px 8px;
  background: white;
  background: rgba(255, 255, 255, 0.8);
  box-shadow: 0 0 15px rgba(0, 0, 0, 0.2);
  border-radius: 5px;
  font-weight: bold;
}
```

```
*/

const CustomControlExample = () => {
  return (
    <MapContainer
      center={[51.505, -0.09]}
      zoom={13}
      style={{ height: '500px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
      />
      <ZoomInfo />
    </MapContainer>
  );
};

export default CustomControlExample;
```

React Component as Control

For more complex controls, you can use React's Portal to render React components as Leaflet controls:

```
import React, { useEffect, useState } from 'react';
import ReactDOM from 'react-dom';
import { createPortal } from 'react-dom';
import { MapContainer, TileLayer, useMap, Marker } from 'react-leaflet';
import L from 'leaflet';

// React component to be used as a control
const SearchControl = ({ onSearch }) => {
  const [searchTerm, setSearchTerm] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    if (searchTerm) {
      onSearch(searchTerm);
    }
  };

  return (
    <div
      className="leaflet-control leaflet-bar"
      style={{ padding: '10px', background: 'white' }}
    >
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          value={searchTerm}
```

```
            onChange={(e) => setSearchTerm(e.target.value)}
            placeholder="Search location..."
            style={{ marginRight: '5px' }}
          />
          <button type="submit">Search</button>
        </form>
      </div>
    );
  };

  // Component to add the React control to the map
  const ReactControlContainer = ({ children }) => {
    const map = useMap();
    const [container, setContainer] = useState(null);

    useEffect(() => {
      // Create a control container
      const controlContainer = L.Control.extend({
        options: {
          position: 'topright',
        },
        onAdd: function () {
          const div = L.DomUtil.create('div', '');
          // Prevent map click events from propagating to the map
          L.DomEvent.disableClickPropagation(div);
          L.DomEvent.disableScrollPropagation(div);
          return div;
        },
      });

      const controlInstance = new controlContainer();
      map.addControl(controlInstance);

      // Set the container for the portal
      setContainer(controlInstance.getContainer());

      return () => {
        map.removeControl(controlInstance);
      };
    }, [map]);

    // Render children into the container using portal
    return container ? createPortal(children, container) : null;
  };

  // Example usage
  const MapWithSearchControl = () => {
    const [marker, setMarker] = useState(null);

    // Dummy search function - in a real app, you would use a geocoding service
    const handleSearch = (searchTerm) => {
      // For demo, just place a marker at a random position near London
      const randomLat = 51.505 + (Math.random() - 0.5) * 0.1;
      const randomLng = -0.09 + (Math.random() - 0.5) * 0.1;
```

```
    setMarker({
      position: [randomLat, randomLng],
      name: searchTerm,
    });
  };

  return (
    <MapContainer
      center={[51.505, -0.09]}
      zoom={13}
      style={{ height: '500px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
      />

      <ReactControlContainer>
        <SearchControl onSearch={handleSearch} />
      </ReactControlContainer>

      {marker && (
        <Marker position={marker.position}>
          <Popup>{marker.name}</Popup>
        </Marker>
      )}
    </MapContainer>
  );
};

export default MapWithSearchControl;
```

Basemaps and Overlay Maps

Let's explore more sophisticated management of basemaps and overlays:

```
import React, { useState } from 'react';
import {
  MapContainer,
  TileLayer,
  LayersControl,
  Marker,
  Popup,
  GeoJSON,
  WMSTileLayer,
} from 'react-leaflet';

const { BaseLayer, Overlay } = LayersControl;
```

```
// Sample GeoJSON data for demonstration
const sampleGeoJSON = {
  type: 'FeatureCollection',
  features: [
    {
      type: 'Feature',
      properties: { name: 'Feature 1' },
      geometry: {
        type: 'Polygon',
        coordinates: [
          [
            [-0.08, 51.5],
            [-0.08, 51.52],
            [-0.06, 51.52],
            [-0.06, 51.5],
            [-0.08, 51.5],
          ],
        ],
      },
    },
    {
      type: 'Feature',
      properties: { name: 'Feature, 2' },
      geometry: {
        type: 'Point',
        coordinates: [-0.1, 51.515],
      },
    },
  ],
};

const AdvancedLayersExample = () => {
  // State for the currently visible basemap
  const [activeBasemap, setActiveBasemap] = useState('osm');

  // Custom style function for GeoJSON
  const geoJSONStyle = (feature) => {
    return {
      color: '#ff7800',
      weight: 2,
      opacity: 0.65,
    };
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h4>Current Basemap: {activeBasemap}</h4>
      </div>

      <MapContainer
        center={[51.505, -0.09]}
        zoom={13}
        style={{ height: '500px', width: '100%' }}
```

```
      >
        <LayersControl position="topright">
          {/* Basemap options */}
          <BaseLayer
            checked
            name="OpenStreetMap"
            onChange={() => setActiveBasemap('osm')}
          >
            <TileLayer
              url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
              attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
            />
          </BaseLayer>

          <BaseLayer
            name="Satellite"
            onChange={() => setActiveBasemap('satellite')}
          >
            <TileLayer

url="https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/
tile/{z}/{y}/{x}"
              attribution='&copy; <a href="https://www.esri.com">Esri</a>,
Sources: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA,
USGS, AeroGRID, IGN, and the GIS User Community'
            />
          </BaseLayer>

          <BaseLayer
            name="Terrain"
            onChange={() => setActiveBasemap('terrain')}
          >
            <TileLayer
              url="https://stamen-tiles-{s}.a.ssl.fastly.net/terrain/{z}/{x}/{y}
{r}.png"
              attribution='Map tiles by <a href="http://stamen.com">Stamen
Design</a>, <a href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a>
&mdash; Map data &copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
              subdomains="abcd"
            />
          </BaseLayer>

          {/* Overlay options */}
          <Overlay name="Points of Interest">
            <LayersControl.Group>
              <Marker position={[51.505, -0.09]}>
                <Popup>A key landmark in London</Popup>
              </Marker>
              <Marker position={[51.51, -0.1]}>
                <Popup>Another interesting place</Popup>
              </Marker>
            </LayersControl.Group>
```

```
            </Overlay>

            <Overlay name="GeoJSON Layer">
              <GeoJSON
                data={sampleGeoJSON}
                style={geoJSONStyle}
                onEachFeature={(feature, layer) => {
                  if (feature.properties && feature.properties.name) {
                    layer.bindPopup(feature.properties.name);
                  }
                }}
              />
            </Overlay>

            <Overlay name="Weather Radar">
              <WMSTileLayer
                url="http://mesonet.agron.iastate.edu/cgi-bin/wms/nexrad/n0r.cgi"
                layers="nexrad-n0r-900913"
                format="image/png"
                transparent={true}
                attribution="Weather data © 2012 IEM Nexrad"
              />
            </Overlay>
          </LayersControl>
        </MapContainer>
      </div>
    );
};

export default AdvancedLayersExample;
```

Best Practices for Layers

1. **Performance**
   - Only load layers that are visible
   - Use clustering for markers (covered in Advanced Features)
   - Consider using vector tiles for complex data
2. **User Experience**
   - Organize layers logically in the layer control
   - Provide clear names and descriptions for layers
   - Ensure layer visibility states persist across user interactions
3. **Code Organization**
   - Extract layer definitions to separate files for complex applications
   - Use context or state management to track layer visibility across components
4. **Common Pitfalls**

- Overloading the map with too many layers, causing performance issues
- Not cleaning up layer event listeners, leading to memory leaks
- Forgetting to handle loading states for layers that fetch data asynchronously

Geometry and Vector Features

Points, Lines, Polygons, and Circles

Vector features are the geometric objects displayed on your map. Let's explore how to create and manipulate different types of vector elements.

Markers (Points)

Markers represent single points on the map:

```
import React from 'react';
import { MapContainer, TileLayer, Marker, Popup } from 'react-leaflet';

const MarkersExample = () => {
  const markers = [
    {
      id: 1,
      position: [51.505, -0.09],
      name: 'London',
      description: 'Capital of England',
    },
    {
      id: 2,
      position: [51.51, -0.1],
      name: "Regent's Park",
      description: 'Beautiful green space',
    },
    {
      id: 3,
      position: [51.49, -0.12],
      name: 'Westminster',
      description: 'Historic political area',
    },
  ];

  return (
    <MapContainer
      center={[51.505, -0.09]}
      zoom={12}
      style={{ height: '500px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
      />
```

```
        {markers.map((marker) => (
          <Marker key={marker.id} position={marker.position}>
            <Popup>
              <div>
                <h3>{marker.name}</h3>
                <p>{marker.description}</p>
              </div>
            </Popup>
          </Marker>
        ))}
      </MapContainer>
    );
};

export default MarkersExample;
```

Polylines (Lines)

Polylines connect multiple points with a line:

```
import React from 'react';
import { MapContainer, TileLayer, Polyline, Popup } from 'react-leaflet';

const PolylinesExample = () => {
  // A single route with multiple points
  const londonTour = [
    [51.505, -0.09], // Start point
    [51.51, -0.1], // Regent's Park
    [51.51, -0.12], // Baker Street
    [51.5, -0.12], // Oxford Street
    [51.49, -0.13], // Piccadilly Circus
    [51.5, -0.14], // Trafalgar Square
    [51.5, -0.12], // Covent Garden
    [51.505, -0.09], // Back to start
  ];

  // Multiple separate routes
  const busRoutes = [
    {
      id: 1,
      name: 'Route 1',
      color: 'red',
      positions: [
        [51.52, -0.09],
        [51.53, -0.11],
        [51.54, -0.13],
      ],
    },
    {
      id: 2,
      name: 'Route 2',
```

```
        color: 'blue',
        positions: [
          [51.49, -0.08],
          [51.48, -0.1],
          [51.47, -0.12],
        ],
      },
    ];

    return (
      <MapContainer
        center={[51.505, -0.09]}
        zoom={12}
        style={{ height: '500px', width: '100%' }}
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        {/* London tour route */}
        <Polyline
          positions={londonTour}
          color="purple"
          weight={3}
          dashArray="5, 10"
        >
          <Popup>London Tour Route</Popup>
        </Polyline>

        {/* Bus routes */}
        {busRoutes.map((route) => (
          <Polyline
            key={route.id}
            positions={route.positions}
            color={route.color}
            weight={3}
          >
            <Popup>{route.name}</Popup>
          </Polyline>
        ))}
      </MapContainer>
    );
};

export default PolylinesExample;
```

Polygons (Areas)

Polygons define enclosed areas:

```
import React from 'react';
import { MapContainer, TileLayer, Polygon, Tooltip } from 'react-leaflet';

const PolygonsExample = () => {
  // Define some polygon coordinates
  const londonParks = [
    {
      id: 1,
      name: "Regent's Park",
      color: '#43a047',
      positions: [
        [51.531, -0.164],
        [51.531, -0.148],
        [51.523, -0.143],
        [51.518, -0.155],
        [51.523, -0.163],
      ],
    },
    {
      id: 2,
      name: 'Hyde Park',
      color: '#388e3c',
      positions: [
        [51.508, -0.175],
        [51.508, -0.151],
        [51.502, -0.149],
        [51.5, -0.173],
      ],
    },
    {
      id: 3,
      name: 'Green Park',
      color: '#4caf50',
      positions: [
        [51.507, -0.145],
        [51.507, -0.139],
        [51.503, -0.139],
        [51.503, -0.144],
      ],
    },
  ];

  // Polygon with a hole (like a donut)
  const polygonWithHole = {
    id: 4,
    name: 'Olympic Park',
    color: '#8bc34a',
    positions: [
      // Outer ring
      [
        [51.547, -0.022],
        [51.547, -0.01],
        [51.538, -0.01],
```

```
          [51.538, -0.022],
        ],
        // Inner hole
        [
          [51.544, -0.018],
          [51.544, -0.014],
          [51.541, -0.014],
          [51.541, -0.018],
        ],
      ],
    };

    return (
      <MapContainer
        center={[51.515, -0.13]}
        zoom={12}
        style={{ height: '500px', width: '100%' }}
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        {/* Regular polygons */}
        {londonParks.map((park) => (
          <Polygon
            key={park.id}
            positions={park.positions}
            pathOptions={{
              color: park.color,
              fillOpacity: 0.6,
              weight: 2,
            }}
          >
            <Tooltip sticky>{park.name}</Tooltip>
          </Polygon>
        ))}

        {/* Polygon with a hole */}
        <Polygon
          positions={polygonWithHole.positions}
          pathOptions={{
            color: polygonWithHole.color,
            fillOpacity: 0.6,
            weight: 2,
          }}
        >
          <Tooltip sticky>{polygonWithHole.name}</Tooltip>
        </Polygon>
      </MapContainer>
    );
  };
```

```
export default PolygonsExample;
```

Circles and CircleMarkers

Circles represent circular areas with a fixed radius:

```
import React from 'react';
import {
  MapContainer,
  TileLayer,
  Circle,
  CircleMarker,
  Popup,
} from 'react-leaflet';

const CirclesExample = () => {
  const circles = [
    {
      id: 1,
      center: [51.505, -0.09],
      radius: 500, // in meters
      color: '#f44336',
      name: 'Coverage Area 1',
    },
    {
      id: 2,
      center: [51.51, -0.1],
      radius: 300, // in meters
      color: '#2196f3',
      name: 'Coverage Area 2',
    },
  ];

  const circleMarkers = [
    {
      id: 1,
      center: [51.49, -0.08],
      radius: 15, // in pixels
      color: '#ff9800',
      name: 'Point of Interest 1',
    },
    {
      id: 2,
      center: [51.485, -0.1],
      radius: 10, // in pixels
      color: '#9c27b0',
      name: 'Point of Interest 2',
    },
  ];
```

```
  return (
    <MapContainer
      center={[51.505, -0.09]}
      zoom={13}
      style={{ height: '500px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
      />

      {/* Circles (fixed radius in meters, scales with zoom) */}
      {circles.map((circle) => (
        <Circle
          key={circle.id}
          center={circle.center}
          radius={circle.radius}
          pathOptions={{
            color: circle.color,
            fillColor: circle.color,
            fillOpacity: 0.3,
          }}
        >
          <Popup>
            <strong>{circle.name}</strong>
            <br />
            Radius: {circle.radius}m
          </Popup>
        </Circle>
      ))}

      {/* CircleMarkers (fixed radius in pixels, doesn't scale with zoom) */}
      {circleMarkers.map((marker) => (
        <CircleMarker
          key={marker.id}
          center={marker.center}
          radius={marker.radius}
          pathOptions={{
            color: marker.color,
            fillColor: marker.color,
            fillOpacity: 0.7,
          }}
        >
          <Popup>
            <strong>{marker.name}</strong>
          </Popup>
        </CircleMarker>
      ))}
    </MapContainer>
  );
};

export default CirclesExample;
```

Key Differences: Circle vs CircleMarker

- **Circle**: Radius is in meters and the circle scales with zoom
- **CircleMarker**: Radius is in pixels and remains the same size regardless of zoom level

Styling Vector Features

Leaflet offers various styling options for vector elements. Let's explore how to style different features:

Path Options

Path options control the appearance of all vector layers:

```
import React, { useState } from 'react';
import {
  MapContainer,
  TileLayer,
  Polygon,
  Polyline,
  Circle,
  Tooltip,
} from 'react-leaflet';

const VectorStylingExample = () => {
  // Sample feature with customizable style
  const [pathOptions, setPathOptions] = useState({
    color: '#3388ff',
    weight: 3,
    opacity: 0.7,
    fillColor: '#3388ff',
    fillOpacity: 0.2,
    dashArray: '',
    lineCap: 'round',
    lineJoin: 'round',
  });

  // Sample geometries
  const polygon = [
    [51.515, -0.13],
    [51.52, -0.12],
    [51.52, -0.11],
    [51.515, -0.1],
    [51.51, -0.12],
  ];

  const polyline = [
    [51.505, -0.13],
    [51.51, -0.12],
    [51.51, -0.1],
    [51.505, -0.09],
```

```
  ];

  // Update style properties
  const updateStyle = (property, value) => {
    setPathOptions({
      ...pathOptions,
      [property]: value,
    });
  };

  // For numeric inputs
  const handleNumericChange = (property, e) => {
    updateStyle(property, parseFloat(e.target.value));
  };

  return (
    <div>
      <div style={{ marginBottom: '15px' }}>
        <h3>Style Options</h3>
        <div style={{ display: 'flex', flexWrap: 'wrap', gap: '10px' }}>
          <div>
            <label>Color: </label>
            <input
              type="color"
              value={pathOptions.color}
              onChange={(e) => updateStyle('color', e.target.value)}
            />
          </div>

          <div>
            <label>Weight: </label>
            <input
              type="range"
              min="1"
              max="10"
              value={pathOptions.weight}
              onChange={(e) => handleNumericChange('weight', e)}
            /> {pathOptions.weight}
          </div>

          <div>
            <label>Opacity: </label>
            <input
              type="range"
              min="0"
              max="1"
              step="0.1"
              value={pathOptions.opacity}
              onChange={(e) => handleNumericChange('opacity', e)}
            /> {pathOptions.opacity}
          </div>

          <div>
            <label>Fill Color: </label>
```

```
            <input
              type="color"
              value={pathOptions.fillColor}
              onChange={(e) => updateStyle('fillColor', e.target.value)}
            />
          </div>

          <div>
            <label>Fill Opacity: </label>
            <input
              type="range"
              min="0"
              max="1"
              step="0.1"
              value={pathOptions.fillOpacity}
              onChange={(e) => handleNumericChange('fillOpacity', e)}
            /> {pathOptions.fillOpacity}
          </div>

          <div>
            <label>Dash Array: </label>
            <select
              value={pathOptions.dashArray}
              onChange={(e) => updateStyle('dashArray', e.target.value)}
            >
              <option value="">Solid</option>
              <option value="5,10">Dashed (5,10)</option>
              <option value="1,5">Dotted (1,5)</option>
              <option value="15,10,5,10">Dash-Dot (15,10,5,10)</option>
            </select>
          </div>
        </div>
      </div>

      <MapContainer
        center={[51.51, -0.11]}
        zoom={13}
        style={{ height: '500px', width: '100%' }}
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        {/* Apply the same styling to different geometry types */}
        <Polygon positions={polygon} pathOptions={pathOptions}>
          <Tooltip>Polygon</Tooltip>
        </Polygon>

        <Polyline positions={polyline} pathOptions={pathOptions}>
          <Tooltip>Polyline</Tooltip>
        </Polyline>
```

```
        <Circle center={[51.505, -0.11]} radius={300} pathOptions={pathOptions}>
          <Tooltip>Circle</Tooltip>
        </Circle>
      </MapContainer>
    </div>
  );
};

export default VectorStylingExample;
```

Dynamic Styling

You can change styles based on data or interactions:

```
import React, { useState } from 'react';
import { MapContainer, TileLayer, Polygon, useMap } from 'react-leaflet';

// Data for districts with some metrics
const districts = [
  {
    id: 1,
    name: 'District A',
    value: 75, // Some metric (like population density)
    coordinates: [
      [51.52, -0.12],
      [51.52, -0.1],
      [51.51, -0.09],
      [51.5, -0.1],
      [51.5, -0.12],
    ],
  },
  {
    id: 2,
    name: 'District B',
    value: 45,
    coordinates: [
      [51.51, -0.14],
      [51.52, -0.13],
      [51.52, -0.12],
      [51.5, -0.12],
      [51.5, -0.13],
    ],
  },
  {
    id: 3,
    name: 'District C',
    value: 90,
    coordinates: [
      [51.5, -0.1],
      [51.51, -0.09],
      [51.51, -0.07],
```

```
        [51.49, -0.07],
        [51.49, -0.09],
      ],
    },
    {
      id: 4,
      name: 'District D',
      value: 30,
      coordinates: [
        [51.5, -0.13],
        [51.5, -0.12],
        [51.49, -0.11],
        [51.48, -0.12],
        [51.49, -0.14],
      ],
    },
  ];

  const DynamicStylingExample = () => {
    const [hoveredId, setHoveredId] = useState(null);
    const [selectedId, setSelectedId] = useState(null);

    // Function to determine color based on value
    const getValueColor = (value) => {
      // Color gradient from red (low) to green (high)
      if (value >= 75) return '#00C853'; // High - green
      if (value >= 50) return '#FFD600'; // Medium - yellow
      return '#FF3D00'; // Low - red
    };

    // Style function for polygons
    const getPolygonStyle = (district) => {
      const isHovered = district.id === hoveredId;
      const isSelected = district.id === selectedId;

      let style = {
        color: '#333',
        weight: isHovered || isSelected ? 3 : 1,
        fillColor: getValueColor(district.value),
        fillOpacity: isSelected ? 0.7 : isHovered ? 0.5 : 0.3,
      };

      if (isSelected) {
        style.dashArray = '3';
      }

      return style;
    };

    // Handle events
    const handleMouseOver = (id) => {
      setHoveredId(id);
    };
```

```
  const handleMouseOut = () => {
    setHoveredId(null);
  };

  const handleClick = (id) => {
    setSelectedId(id === selectedId ? null : id);
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>District Information</h3>
        {selectedId ? (
          <div>
            <p>
              <strong>
                Selected: {districts.find((d) => d.id === selectedId)?.name}
              </strong>
              <br />
              Value: {districts.find((d) => d.id === selectedId)?.value}
            </p>
          </div>
        ) : (
          <p>Click on a district to view details</p>
        )}

        <div style={{ display: 'flex', marginTop: '10px' }}>
          <div
            style={{
              display: 'flex',
              alignItems: 'center',
              marginRight: '20px',
            }}
          >
            <div
              style={{
                backgroundColor: '#FF3D00',
                width: '20px',
                height: '20px',
                marginRight: '5px',
              }}
            ></div>
            <span>Low Value</span>
          </div>

          <div
            style={{
              display: 'flex',
              alignItems: 'center',
              marginRight: '20px',
            }}
          >
            <div
              style={{
```

```
                backgroundColor: '#FFD600',
                width: '20px',
                height: '20px',
                marginRight: '5px',
              }}
            ></div>
            <span>Medium Value</span>
          </div>

          <div
            style={{
              display: 'flex',
              alignItems: 'center',
            }}
          >
            <div
              style={{
                backgroundColor: '#00C853',
                width: '20px',
                height: '20px',
                marginRight: '5px',
              }}
            ></div>
            <span>High Value</span>
          </div>
        </div>
      </div>

      <MapContainer
        center={[51.505, -0.11]}
        zoom={13}
        style={{ height: '500px', width: '100%' }}
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        {districts.map((district) => (
          <Polygon
            key={district.id}
            positions={district.coordinates}
            pathOptions={getPolygonStyle(district)}
            eventHandlers={{
              mouseover: () => handleMouseOver(district.id),
              mouseout: handleMouseOut,
              click: () => handleClick(district.id),
            }}
          >
            <Tooltip>
              <div>
                <strong>{district.name}</strong>
                <br />
```

```
                Value: {district.value}
              </div>
            </Tooltip>
          </Polygon>
        ))}
      </MapContainer>
    </div>
  );
};

export default DynamicStylingExample;
```

Style Options Reference

Common style properties for paths:

- **color**: Stroke color
- **weight**: Stroke width in pixels
- **opacity**: Stroke opacity
- **fillColor**: Fill color (for closed shapes)
- **fillOpacity**: Fill opacity (for closed shapes)
- **dashArray**: Pattern of dashes and gaps (e.g., "5, 10")
- **lineCap**: Shape of line endings ('butt', 'round', 'square')
- **lineJoin**: Shape of line joints ('miter', 'round', 'bevel')
- **fillRule**: How to fill a shape ('evenodd', 'nonzero')
- **renderer**: Custom renderer to use for the layer
- **className**: Custom class name for the element

GeoJSON Integration

GeoJSON is a standard format for representing geographical features. Leaflet has excellent support for GeoJSON, making it easy to display complex geographical data.

Basic GeoJSON Example

```
import React from 'react';
import { MapContainer, TileLayer, GeoJSON } from 'react-leaflet';

// Sample GeoJSON data (simplified)
const geoJsonData = {
  type: 'FeatureCollection',
  features: [
    {
      type: 'Feature',
      properties: {
        name: 'Central Park',
        type: 'park',
      },
      geometry: {
```

```
          type: 'Polygon',
          coordinates: [
            [
              [-73.9812, 40.7681],
              [-73.9732, 40.7681],
              [-73.9732, 40.7645],
              [-73.9812, 40.7645],
              [-73.9812, 40.7681],
            ],
          ],
        },
      },
      {
        type: 'Feature',
        properties: {
          name: 'Empire State Building',
          type: 'landmark',
        },
        geometry: {
          type: 'Point',
          coordinates: [-73.9857, 40.7484],
        },
      },
      {
        type: 'Feature',
        properties: {
          name: 'Broadway',
          type: 'road',
        },
        geometry: {
          type: 'LineString',
          coordinates: [
            [-73.9875, 40.7577],
            [-73.9867, 40.756],
            [-73.986, 40.7545],
            [-73.9852, 40.7532],
          ],
        },
      },
    ],
  };

  const BasicGeoJSONExample = () => {
    // Style function based on feature properties
    const geoJSONStyle = (feature) => {
      const featureType = feature.properties.type;

      switch (featureType) {
        case 'park':
          return {
            fillColor: '#4CAF50',
            color: '#388E3C',
            weight: 2,
            fillOpacity: 0.5,
```

```
        };
      case 'landmark':
        return {
          color: '#F44336',
          weight: 2,
          fillColor: '#EF5350',
          fillOpacity: 0.7,
          radius: 8, // For CircleMarker for points
        };
      case 'road':
        return {
          color: '#FF9800',
          weight: 3,
          opacity: 0.7,
        };
      default:
        return {
          color: '#2196F3',
          weight: 2,
          fillColor: '#64B5F6',
          fillOpacity: 0.5,
        };
    }
  };

  // Function to handle each feature
  const onEachFeature = (feature, layer) => {
    // Bind popup with feature information
    if (feature.properties && feature.properties.name) {
      layer.bindPopup(`
        <strong>${feature.properties.name}</strong><br>
        Type: ${feature.properties.type}
      `);
    }

    // Add event handlers
    layer.on({
      mouseover: (e) => {
        const layer = e.target;
        layer.setStyle({
          weight: 4,
          fillOpacity: 0.8,
        });
      },
      mouseout: (e) => {
        const layer = e.target;
        layer.setStyle(geoJSONStyle(feature));
      },
    });
  };

  // Function to handle point features
  const pointToLayer = (feature, latlng) => {
    return L.circleMarker(latlng, geoJSONStyle(feature));
```

```
  };

  return (
    <MapContainer
      center={[40.75, -73.98]}
      zoom={13}
      style={{ height: '500px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
      />

      <GeoJSON
        data={geoJsonData}
        style={geoJSONStyle}
        onEachFeature={onEachFeature}
        pointToLayer={pointToLayer}
      />
    </MapContainer>
  );
};

export default BasicGeoJSONExample;
```

Loading External GeoJSON

In real applications, you'll often load GeoJSON from an external source:

```
import React, { useState, useEffect } from 'react';
import {
  MapContainer,
  TileLayer,
  GeoJSON,
  LayersControl,
  Marker,
  Popup,
} from 'react-leaflet';
import L from 'leaflet';

const { BaseLayer, Overlay } = LayersControl;

const ExternalGeoJSONExample = () => {
  const [geoJsonData, setGeoJsonData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    // Fetch GeoJSON data
    const fetchGeoJson = async () => {
```

```
        try {
          setLoading(true);
          // Example URL - replace with your actual GeoJSON endpoint
          const response = await fetch(
            'https://data.nasa.gov/api/geospatial/7zbq-j77a?
method=export&format=GeoJSON'
          );

          if (!response.ok) {
            throw new Error(`Failed to fetch GeoJSON: ${response.status}`);
          }

          const data = await response.json();
          setGeoJsonData(data);
          setLoading(false);
        } catch (err) {
          console.error('Error fetching GeoJSON:', err);
          setError(err.message);
          setLoading(false);
        }
      };

      fetchGeoJson();
    }, []);

    // Style function
    const geoJSONStyle = () => {
      return {
        color: '#3388ff',
        weight: 2,
        fillOpacity: 0.3,
      };
    };

    // Function to handle each feature
    const onEachFeature = (feature, layer) => {
      if (feature.properties) {
        // Create a popup with feature properties
        const popupContent = Object.keys(feature.properties)
          .map((key) => `<strong>${key}:</strong> ${feature.properties[key]}`)
          .join('<br>');

        layer.bindPopup(popupContent);
      }
    };

    return (
      <div>
        {loading && <div>Loading GeoJSON data...</div>}
        {error && <div>Error: {error}</div>}

        <MapContainer
          center={[0, 0]}
          zoom={2}
```

```
        style={{ height: '500px', width: '100%' }}
      >
        <LayersControl position="topright">
          <BaseLayer checked name="OpenStreetMap">
            <TileLayer
              url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
              attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
            />
          </BaseLayer>

          <BaseLayer name="Satellite">
            <TileLayer

url="https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/
tile/{z}/{y}/{x}"
              attribution='&copy; <a href="https://www.esri.com">Esri</a>'
            />
          </BaseLayer>

          {!loading && !error && geoJsonData && (
            <Overlay checked name="GeoJSON Data">
              <GeoJSON
                data={geoJsonData}
                style={geoJSONStyle}
                onEachFeature={onEachFeature}
              />
            </Overlay>
          )}
        </LayersControl>
      </MapContainer>
    </div>
  );
};

export default ExternalGeoJSONExample;
```

Handling Large GeoJSON Files

Large GeoJSON files can cause performance issues. Here are some strategies to handle them:

```
import React, { useState, useEffect, useMemo } from 'react';
import {
  MapContainer,
  TileLayer,
  GeoJSON,
  LayersControl,
  ZoomControl,
} from 'react-leaflet';

const LargeGeoJSONExample = () => {
```

```
  const [geoJsonData, setGeoJsonData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [simplificationLevel, setSimplificationLevel] = useState('medium');

  // Fetch GeoJSON data
  useEffect(() => {
    const fetchGeoJson = async () => {
      try {
        setLoading(true);
        // Replace with your actual GeoJSON URL
        const response = await fetch(
          'https://your-geojson-url.com/data.geojson'
        );

        if (!response.ok) {
          throw new Error(`Failed to fetch GeoJSON: ${response.status}`);
        }

        const data = await response.json();
        setGeoJsonData(data);
        setLoading(false);
      } catch (err) {
        console.error('Error fetching GeoJSON:', err);
        setError(err.message);
        setLoading(false);
      }
    };

    fetchGeoJson();
  }, []);

  // Simplify GeoJSON based on selected level
  // In a real app, you might use a library like Turf.js for this
  const simplifiedGeoJson = useMemo(() => {
    if (!geoJsonData) return null;

    // Simple mock implementation - in a real app use proper simplification
    // like Turf.js's turf.simplify() function
    const simplifyGeometry = (geometry, level) => {
      if (!geometry || !geometry.coordinates) return geometry;

      // Simplification factors for different levels
      const factors = {
        high: 1, // No simplification
        medium: 2, // Keep every 2nd point
        low: 4, // Keep every 4th point
      };

      const factor = factors[level] || 1;

      // Apply simplification based on geometry type
      if (geometry.type === 'Point' || geometry.type === 'MultiPoint') {
        return geometry; // Don't simplify points
```

```
      } else if (geometry.type === 'LineString') {
        return {
          ...geometry,
          coordinates: geometry.coordinates.filter((_, i) => i % factor === 0),
        };
      } else if (
        geometry.type === 'Polygon' ||
        geometry.type === 'MultiPolygon'
      ) {
        // Simplify each ring in the polygon
        return {
          ...geometry,
          coordinates: geometry.coordinates.map((ring) =>
            ring.filter(
              (_, i) => i % factor === 0 || i === 0 || i === ring.length - 1
            )
          ),
        };
      }

      return geometry;
    };

    // Create a copy of the GeoJSON with simplified geometries
    return {
      ...geoJsonData,
      features: geoJsonData.features.map((feature) => ({
        ...feature,
        geometry: simplifyGeometry(feature.geometry, simplificationLevel),
      })),
    };
  }, [geoJsonData, simplificationLevel]);

  // Style function with reduced detail at lower zoom levels
  const geoJSONStyle = (feature) => {
    return {
      color: '#3388ff',
      weight: 2,
      fillOpacity: 0.3,
    };
  };

  // Add features only at appropriate zoom levels
  const filter = (feature, layer) => {
    // Example: Only show large features at lower zoom levels
    if (feature.properties && feature.properties.area) {
      // This would depend on the specific properties in your GeoJSON
      return feature.properties.area > 1000000;
    }
    return true;
  };

  return (
    <div>
```

```
        <div style={{ marginBottom: '10px' }}>
          <h3>GeoJSON Simplification Level</h3>
          <div>
            <label>
              <input
                type="radio"
                value="high"
                checked={simplificationLevel === 'high'}
                onChange={() => setSimplificationLevel('high')}
              />{' '}
              High Detail
            </label>
            <label style={{ marginLeft: '10px' }}>
              <input
                type="radio"
                value="medium"
                checked={simplificationLevel === 'medium'}
                onChange={() => setSimplificationLevel('medium')}
              />{' '}
              Medium Detail
            </label>
            <label style={{ marginLeft: '10px' }}>
              <input
                type="radio"
                value="low"
                checked={simplificationLevel === 'low'}
                onChange={() => setSimplificationLevel('low')}
              />{' '}
              Low Detail
            </label>
          </div>
        </div>

        {loading && <div>Loading GeoJSON data...</div>}
        {error && <div>Error: {error}</div>}

        <MapContainer
          center={[0, 0]}
          zoom={2}
          style={{ height: '500px', width: '100%' }}
          zoomControl={false}
        >
          <ZoomControl position="topright" />
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />

          {!loading && !error && simplifiedGeoJson && (
            <GeoJSON
              key={simplificationLevel} // Force re-render when simplification
changes
              data={simplifiedGeoJson}
```

```
              style={geoJSONStyle}
              filter={filter}
            />
          )}
        </MapContainer>

        <div style={{ marginTop: '10px' }}>
          <p>
            <strong>Tips for handling large GeoJSON:</strong>
          </p>
          <ul>
            <li>Simplify geometries based on zoom level</li>
            <li>
              Filter features to show only those relevant at the current zoom
            </li>
            <li>Use clustering for point data (see Advanced Features section)</li>
            <li>
              Consider using vector tiles instead of raw GeoJSON for very large
              datasets
            </li>
            <li>
              Pre-process GeoJSON on the server to reduce size before sending to
              client
            </li>
          </ul>
        </div>
      </div>
    );
};

export default LargeGeoJSONExample;
```

Custom Markers and Icons

Default Leaflet markers can be replaced with custom icons:

```
import React from 'react';
import { MapContainer, TileLayer, Marker, Popup } from 'react-leaflet';
import L from 'leaflet';

// Define custom icons
const createCustomIcon = (
  iconUrl,
  iconSize = [25, 41],
  iconAnchor = [12, 41]
) => {
  return L.icon({
    iconUrl,
    iconSize: iconSize,
    iconAnchor: iconAnchor,
    popupAnchor: [0, -iconAnchor[1]],
```

```
    });
  };

  const CustomMarkersExample = () => {
    // Create icon instances
    const icons = {
      restaurant: createCustomIcon(
        'https://cdn-icons-png.flaticon.com/512/562/562678.png',
        [32, 32],
        [16, 32]
      ),
      hotel: createCustomIcon(
        'https://cdn-icons-png.flaticon.com/512/1475/1475996.png',
        [32, 32],
        [16, 32]
      ),
      attraction: createCustomIcon(
        'https://cdn-icons-png.flaticon.com/512/1141/1141117.png',
        [32, 32],
        [16, 32]
      ),
      shopping: createCustomIcon(
        'https://cdn-icons-png.flaticon.com/512/1170/1170678.png',
        [32, 32],
        [16, 32]
      ),
    };

    // Sample POI data
    const points = [
      {
        id: 1,
        position: [51.505, -0.09],
        name: 'Fancy Restaurant',
        type: 'restaurant',
        description: 'Fine dining experience',
      },
      {
        id: 2,
        position: [51.51, -0.1],
        name: 'Grand Hotel',
        type: 'hotel',
        description: '5-star luxury accommodation',
      },
      {
        id: 3,
        position: [51.5, -0.11],
        name: 'Historic Museum',
        type: 'attraction',
        description: 'Cultural landmark with artifacts',
      },
      {
        id: 4,
        position: [51.495, -0.08],
```

```
      name: 'Shopping Mall',
      type: 'shopping',
      description: 'Retail therapy destination',
    },
    {
      id: 5,
      position: [51.502, -0.07],
      name: 'Local Eatery',
      type: 'restaurant',
      description: 'Authentic local cuisine',
    },
  ];

  // Define custom icon by marker type
  const getMarkerIcon = (type) => {
    return icons[type] || icons.attraction; // Default fallback
  };

  return (
    <MapContainer
      center={[51.505, -0.09]}
      zoom={13}
      style={{ height: '500px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
      />

      {points.map((point) => (
        <Marker
          key={point.id}
          position={point.position}
          icon={getMarkerIcon(point.type)}
        >
          <Popup>
            <div>
              <h3>{point.name}</h3>
              <p>
                <strong>Type:</strong> {point.type}
              </p>
              <p>{point.description}</p>
            </div>
          </Popup>
        </Marker>
      ))}
    </MapContainer>
  );
};

export default CustomMarkersExample;
```

Using SVG Markers

SVG markers offer more flexibility for custom styling:

```
import React from 'react';
import { MapContainer, TileLayer, Marker, Popup } from 'react-leaflet';
import L from 'leaflet';

// Helper function to create SVG icon
const createSvgIcon = (color, size = 36) => {
  // Create an SVG string
  const svgString = `
    <svg xmlns="http://www.w3.org/2000/svg" width="${size}" height="${size}"
viewBox="0 0 24 24" fill="none" stroke="${color}" stroke-width="2" stroke-
linecap="round" stroke-linejoin="round">
      <path d="M21 10c0 7-9 13-9 13s-9-6-9-13a9 9 0 0 1 18 0z"></path>
      <circle cx="12" cy="10" r="3"></circle>
    </svg>
  `;

  // Create a data URL from the SVG
  const svgUrl = `${btoa(svgString)}`;

  // Create and return the icon
  return L.icon({
    iconUrl: svgUrl,
    iconSize: [size, size],
    iconAnchor: [size / 2, size],
    popupAnchor: [0, -size],
  });
};

const SVGMarkersExample = () => {
  // Sample data with different categories and colors
  const points = [
    {
      id: 1,
      position: [51.505, -0.09],
      name: 'Point A',
      category: 'cat1',
      color: '#E53935',
    },
    {
      id: 2,
      position: [51.51, -0.1],
      name: 'Point B',
      category: 'cat2',
      color: '#43A047',
    },
    {
      id: 3,
      position: [51.5, -0.11],
```

```
      name: 'Point C',
      category: 'cat3',
      color: '#1E88E5',
    },
    {
      id: 4,
      position: [51.495, -0.08],
      name: 'Point D',
      category: 'cat1',
      color: '#E53935',
    },
    {
      id: 5,
      position: [51.502, -0.07],
      name: 'Point E',
      category: 'cat2',
      color: '#43A047',
    },
  ];

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>SVG Marker Legend</h3>
        <div style={{ display: 'flex', gap: '15px' }}>
          <div style={{ display: 'flex', alignItems: 'center' }}>
            <div
              style={{
                backgroundColor: '#E53935',
                width: '20px',
                height: '20px',
                marginRight: '5px',
              }}
            ></div>
            <span>Category 1</span>
          </div>
          <div style={{ display: 'flex', alignItems: 'center' }}>
            <div
              style={{
                backgroundColor: '#43A047',
                width: '20px',
                height: '20px',
                marginRight: '5px',
              }}
            ></div>
            <span>Category 2</span>
          </div>
          <div style={{ display: 'flex', alignItems: 'center' }}>
            <div
              style={{
                backgroundColor: '#1E88E5',
                width: '20px',
                height: '20px',
                marginRight: '5px',
```

```
              }}
            ></div>
            <span>Category 3</span>
          </div>
        </div>
      </div>

      <MapContainer
        center={[51.505, -0.09]}
        zoom={13}
        style={{ height: '500px', width: '100%' }}
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        {points.map((point) => (
          <Marker
            key={point.id}
            position={point.position}
            icon={createSvgIcon(point.color)}
          >
            <Popup>
              <div>
                <h3>{point.name}</h3>
                <p>Category: {point.category}</p>
              </div>
            </Popup>
          </Marker>
        ))}
      </MapContainer>
    </div>
  );
};

export default SVGMarkersExample;
```

Dynamic Icon Size by Zoom Level

You can adjust icon size based on zoom level for better user experience:

```
import React, { useState, useEffect, useCallback } from 'react';
import { MapContainer, TileLayer, Marker, Popup, useMap } from 'react-leaflet';
import L from 'leaflet';

// Component to listen for zoom changes
const ZoomListener = ({ onZoomChange }) => {
  const map = useMap();
```

```
  useEffect(() => {
    const handleZoom = () => {
      onZoomChange(map.getZoom());
    };

    map.on('zoom', handleZoom);
    // Initial zoom level
    onZoomChange(map.getZoom());

    return () => {
      map.off('zoom', handleZoom);
    };
  }, [map, onZoomChange]);

  return null;
};

const DynamicIconSizeExample = () => {
  const [zoomLevel, setZoomLevel] = useState(13); // Default zoom

  // Sample data
  const points = [
    { id: 1, position: [51.505, -0.09], name: 'Point A', importance: 5 },
    { id: 2, position: [51.51, -0.1], name: 'Point B', importance: 3 },
    { id: 3, position: [51.5, -0.11], name: 'Point C', importance: 8 },
    { id: 4, position: [51.495, -0.08], name: 'Point D', importance: 2 },
    { id: 5, position: [51.502, -0.07], name: 'Point E', importance: 7 },
  ];

  // Calculate icon size based on zoom level and importance
  const getIconSize = useCallback(
    (importance) => {
      // Base size that scales with zoom
      const baseSize = Math.max(16, Math.min(48, zoomLevel * 2));

      // Adjust for importance (1-10 scale)
      return baseSize * (0.7 + (importance / 10) * 0.6);
    },
    [zoomLevel]
  );

  // Create an SVG icon with dynamic size
  const createDynamicIcon = useCallback(
    (importance) => {
      const size = getIconSize(importance);
      const color =
        importance > 5 ? '#E53935' : importance > 3 ? '#FFB300' : '#43A047';

      // Create an SVG string
      const svgString = `
      <svg xmlns="http://www.w3.org/2000/svg" width="${size}" height="${size}"
viewBox="0 0 24 24" fill="${color}" stroke="#000" stroke-width="1" stroke-
linecap="round" stroke-linejoin="round">
        <circle cx="12" cy="12" r="10"></circle>
```

```
        <text x="12" y="16" text-anchor="middle" font-size="12" fill="white" font-
weight="bold">${importance}</text>
      </svg>
    `;

      // Create a data URL from the SVG
      const svgUrl = `${btoa(svgString)}`;

      // Create and return the icon
      return L.icon({
        iconUrl: svgUrl,
        iconSize: [size, size],
        iconAnchor: [size / 2, size / 2],
        popupAnchor: [0, -size / 2],
      });
    },
    [getIconSize]
  );

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Current Zoom Level: {zoomLevel}</h3>
        <p>
          Notice how the markers change size as you zoom in and out. More
          important points (higher numbers) are larger and colored differently.
        </p>
      </div>

      <MapContainer
        center={[51.505, -0.09]}
        zoom={13}
        style={{ height: '500px', width: '100%' }}
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        <ZoomListener onZoomChange={setZoomLevel} />

        {points.map((point) => (
          <Marker
            key={point.id}
            position={point.position}
            icon={createDynamicIcon(point.importance)}
          >
            <Popup>
              <div>
                <h3>{point.name}</h3>
                <p>Importance: {point.importance}/10</p>
              </div>
            </Popup>
```

```
            </Marker>
          ))}
        </MapContainer>
      </div>
    );
};

export default DynamicIconSizeExample;
```

Common Pitfalls with Custom Icons

1. **Missing icon images**
    - Always verify that icon images can be loaded correctly
    - Consider using inline SVG where possible to avoid external dependencies
    - Use fallback icons if the primary icon fails to load
2. **Incorrect anchor points**
    - The `iconAnchor` property defines where the icon connects to the map coordinates
    - Incorrectly set anchors can cause icons to appear in the wrong places
    - For marker-style icons, the anchor is typically at the bottom center
3. **Sizing issues across zoom levels**
    - Without dynamic sizing, icons may look too large or too small at different zoom levels
    - Consider using different icons at different zoom ranges
4. **Performance with many markers**
    - Creating custom icons for hundreds or thousands of markers can impact performance
    - Consider using marker clustering (covered in the Advanced Features section)
    - For large datasets, reuse icon instances rather than creating new ones for each marker

Tooltips and Popups

Tooltips and popups are essential for displaying additional information about map elements.

Basic Popups and Tooltips

```
import React from 'react';
import {
  MapContainer,
  TileLayer,
  Marker,
  Popup,
  Tooltip,
  Circle,
  Polygon,
} from 'react-leaflet';
```

```
const PopupsAndTooltipsExample = () => {
  // Sample polygon data
  const polygon = [
    [51.515, -0.09],
    [51.52, -0.08],
    [51.52, -0.06],
    [51.51, -0.05],
    [51.505, -0.07],
  ];

  return (
    <MapContainer
      center={[51.505, -0.09]}
      zoom={13}
      style={{ height: '500px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
      />

      {/* Marker with basic popup */}
      <Marker position={[51.505, -0.09]}>
        <Popup>
          <div>
            <h3>Basic Popup</h3>
            <p>This is a simple popup attached to a marker.</p>
          </div>
        </Popup>
      </Marker>

      {/* Marker with tooltip */}
      <Marker position={[51.51, -0.1]}>
        <Tooltip>A simple tooltip</Tooltip>
        <Popup>
          <div>
            <h3>Marker with Tooltip</h3>
            <p>This marker has both a tooltip and a popup.</p>
            <p>
              The tooltip appears on hover, while the popup appears on click.
            </p>
          </div>
        </Popup>
      </Marker>

      {/* Circle with permanent tooltip */}
      <Circle
        center={[51.49, -0.08]}
        radius={400}
        pathOptions={{ color: 'green' }}
      >
        <Tooltip permanent>
```

```
            <span>Permanent tooltip</span>
          </Tooltip>
          <Popup>
            <div>
              <h3>Circle Popup</h3>
              <p>Popups can be attached to any Leaflet layer.</p>
            </div>
          </Popup>
        </Circle>

        {/* Polygon with sticky tooltip */}
        <Polygon positions={polygon} pathOptions={{ color: 'purple' }}>
          <Tooltip sticky>
            <span>Sticky tooltip follows mouse</span>
          </Tooltip>
          <Popup>
            <div>
              <h3>Polygon Area</h3>
              <p>This popup is attached to a polygon.</p>
            </div>
          </Popup>
        </Polygon>
      </MapContainer>
    );
};

export default PopupsAndTooltipsExample;
```

Customizing Popups and Tooltips

You can customize both the appearance and behavior of popups and tooltips:

```
import React, { useState } from 'react';
import {
  MapContainer,
  TileLayer,
  Marker,
  Popup,
  Tooltip,
  useMapEvents,
} from 'react-leaflet';
import './CustomPopupStyles.css'; // You'll need to create this CSS file

// Sample CSS (CustomPopupStyles.css):
/*
.custom-popup .leaflet-popup-content-wrapper {
  background: rgba(0, 0, 0, 0.8);
  color: #fff;
  border-radius: 5px;
  font-family: 'Arial', sans-serif;
}
```

```
.custom-popup .leaflet-popup-tip {
  background: rgba(0, 0, 0, 0.8);
}

.custom-tooltip {
  background: rgba(74, 20, 140, 0.8);
  border: none;
  color: white;
  font-weight: bold;
  border-radius: 2px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);
}
*/

const CustomPopupsAndTooltipsExample = () => {
  const [clickedPosition, setClickedPosition] = useState(null);

  // Sample data
  const locations = [
    {
      id: 1,
      position: [51.505, -0.09],
      name: 'Location A',
      description: 'Important historic site',
    },
    {
      id: 2,
      position: [51.51, -0.1],
      name: 'Location B',
      description: 'Popular tourist attraction',
    },
    {
      id: 3,
      position: [51.49, -0.08],
      name: 'Location C',
      description: 'Local restaurant district',
    },
  ];

  // Map click handler component
  const MapClickHandler = () => {
    useMapEvents({
      click: (e) => {
        setClickedPosition(e.latlng);
      },
    });
    return null;
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Customized Popups and Tooltips</h3>
```

```
          <p>Click anywhere on the map to create a popup at that location.</p>
        </div>

        <MapContainer
          center={[51.505, -0.09]}
          zoom={13}
          style={{ height: '500px', width: '100%' }}
        >
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />

          <MapClickHandler />

          {/* Display markers with custom popups */}
          {locations.map((location) => (
            <Marker key={location.id} position={location.position}>
              <Popup
                className="custom-popup"
                closeButton={false}
                minWidth={200}
                maxWidth={300}
              >
                <div>
                  <h3>{location.name}</h3>
                  <p>{location.description}</p>
                  <button
                    onClick={() => alert(`You clicked on ${location.name}`)}
                  >
                    More Info
                  </button>
                </div>
              </Popup>
              <Tooltip
                className="custom-tooltip"
                direction="top"
                offset={[0, -20]}
                opacity={0.9}
              >
                {location.name}
              </Tooltip>
            </Marker>
          ))}

          {/* Popup at clicked position */}
          {clickedPosition && (
            <Popup
              position={[clickedPosition.lat, clickedPosition.lng]}
              closeOnClick={false}
              onClose={() => setClickedPosition(null)}
              className="custom-popup"
            >
```

```
            <div>
              <h3>Clicked Position</h3>
              <p>Latitude: {clickedPosition.lat.toFixed(6)}</p>
              <p>Longitude: {clickedPosition.lng.toFixed(6)}</p>
            </div>
          </Popup>
        )}
      </MapContainer>
    </div>
  );
};

export default CustomPopupsAndTooltipsExample;
```

Popup and Tooltip Options

Popup Options:

- `className`: Custom CSS class name for the popup
- `pane`: Map pane where the popup will be added
- `closeButton`: Whether to show the close button
- `autoClose`: Whether to close the popup when another popup is opened
- `closeOnClick`: Whether to close the popup when the map is clicked
- `closeOnEscapeKey`: Whether to close the popup when the escape key is pressed
- `offset`: Offset of the popup position relative to the element it's attached to
- `maxWidth`: Maximum width of the popup
- `minWidth`: Minimum width of the popup
- `maxHeight`: Maximum height of the popup

Tooltip Options:

- `className`: Custom CSS class name for the tooltip
- `direction`: Direction of the tooltip relative to its position (`auto`, `top`, `bottom`, `left`, `right`, `center`)
- `permanent`: Whether the tooltip should be permanently visible
- `sticky`: Whether the tooltip should follow the mouse cursor
- `opacity`: Opacity of the tooltip container
- `offset`: Pixel offset of the tooltip position

Dynamically Managing Popups

Sometimes you need to programmatically open or close popups based on user interactions:

```
import React, { useRef, useState } from 'react';
import { MapContainer, TileLayer, Marker, Popup, useMap } from 'react-leaflet';

// Component to control popups programmatically
const PopupController = ({ markers, activeId }) => {
  const map = useMap();
```

```
  // Effect to open popup on activeId change
  React.useEffect(() => {
    if (activeId !== null) {
      const marker = markers.find((m) => m.id === activeId);
      if (marker && marker.ref && marker.ref.current) {
        // Open popup for this marker
        marker.ref.current.openPopup();

        // Pan to the marker
        map.panTo(marker.position);
      }
    }
  }, [activeId, markers, map]);

  return null;
};

const DynamicPopupsExample = () => {
  // Sample data with refs
  const markers = [
    { id: 1, position: [51.505, -0.09], name: 'Marker A', ref: useRef() },
    { id: 2, position: [51.51, -0.1], name: 'Marker B', ref: useRef() },
    { id: 3, position: [51.49, -0.08], name: 'Marker C', ref: useRef() },
    { id: 4, position: [51.495, -0.07], name: 'Marker D', ref: useRef() },
  ];

  // Track which marker is active
  const [activeMarkerId, setActiveMarkerId] = useState(null);

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Dynamic Popup Management</h3>
        <p>Click the buttons to open specific popups:</p>
        <div style={{ display: 'flex', gap: '10px', marginBottom: '10px' }}>
          {markers.map((marker) => (
            <button
              key={marker.id}
              onClick={() => setActiveMarkerId(marker.id)}
            >
              Open {marker.name}
            </button>
          ))}
          <button onClick={() => setActiveMarkerId(null)}>Close All</button>
        </div>
      </div>

      <MapContainer
        center={[51.505, -0.09]}
        zoom={13}
        style={{ height: '500px', width: '100%' }}
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
```

```
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        {/* Render markers with refs */}
        {markers.map((marker) => (
          <Marker key={marker.id} position={marker.position} ref={marker.ref}>
            <Popup>
              <div>
                <h3>{marker.name}</h3>
                <p>
                  This popup was{' '}
                  {activeMarkerId === marker.id
                    ? 'programmatically opened'
                    : 'opened by clicking the marker'}
                  .
                </p>
              </div>
            </Popup>
          </Marker>
        ))}

        {/* Controller to manage popups */}
        <PopupController markers={markers} activeId={activeMarkerId} />
      </MapContainer>
    </div>
  );
};

export default DynamicPopupsExample;
```

Best Practices for Popups and Tooltips

1. **Content Organization**
   - Keep popup content concise and focused
   - Use tooltips for brief information, popups for more detailed content
   - Consider using tabs or collapsible sections for complex popup content
2. **Performance**
   - Avoid loading heavy content like images in tooltips that appear on hover
   - For complex popups, consider loading content dynamically when opened
   - Be cautious with permanent tooltips when you have many markers
3. **User Experience**
   - Make popups easy to close (clear close buttons)
   - Ensure popup content is readable (sufficient contrast, appropriate font size)
   - Consider mobile users (touch-friendly buttons, appropriate sizing)
4. **Styling**

- Use consistent styling across your popups and tooltips
- Test popup styles across different browsers and devices
- Consider custom animations or transitions for smoother interactions

Interactive Features

Handling Map Events

Leaflet provides numerous events that you can listen to for creating interactive experiences.

Basic Map Events

```
import React, { useState } from 'react';
import {
  MapContainer,
  TileLayer,
  useMapEvents,
  Marker,
  Popup,
} from 'react-leaflet';

const BasicMapEventsExample = () => {
  const [eventLog, setEventLog] = useState([]);
  const [clickedPoints, setClickedPoints] = useState([]);

  // Add an event to the log
  const logEvent = (eventName, data = {}) => {
    const timestamp = new Date().toLocaleTimeString();
    setEventLog((prev) => [
      { timestamp, eventName, data },
      ...prev.slice(0, 9), // Keep only the 10 most recent events
    ]);
  };

  // Component that listens to map events
  const MapEventLogger = () => {
    const map = useMapEvents({
      // Mouse events
      click: (e) => {
        logEvent('click', {
          lat: e.latlng.lat.toFixed(6),
          lng: e.latlng.lng.toFixed(6),
        });
        // Add clicked point to the map
        setClickedPoints((prev) => [...prev, e.latlng]);
      },
      dblclick: (e) => {
        logEvent('dblclick', {
          lat: e.latlng.lat.toFixed(6),
          lng: e.latlng.lng.toFixed(6),
        });
      },
```

```
    mousedown: () => logEvent('mousedown'),
    mouseup: () => logEvent('mouseup'),
    mouseover: () => logEvent('mouseover'),
    mouseout: () => logEvent('mouseout'),
    mousemove: (e) => {
      // Don't log every mousemove to avoid flooding the log
      // But we could use this for coordinates display
    },
    contextmenu: (e) => {
      logEvent('contextmenu', {
        lat: e.latlng.lat.toFixed(6),
        lng: e.latlng.lng.toFixed(6),
      });
      e.originalEvent.preventDefault(); // Prevent default context menu
    },

    // Map state events
    zoom: () => {
      logEvent('zoom', { newZoom: map.getZoom() });
    },
    zoomstart: () => logEvent('zoomstart'),
    zoomend: () => logEvent('zoomend'),
    move: () => {
      // Don't log every move to avoid flooding
    },
    movestart: () => logEvent('movestart'),
    moveend: () => {
      const center = map.getCenter();
      logEvent('moveend', {
        center: `${center.lat.toFixed(6)}, ${center.lng.toFixed(6)}`,
        bounds: map.getBounds().toBBoxString(),
      });
    },

    // Layer events
    layeradd: (e) => logEvent('layeradd', { layerId: e.layer._leaflet_id }),
    layerremove: (e) =>
      logEvent('layerremove', { layerId: e.layer._leaflet_id }),

    // Location events
    locationfound: (e) => {
      logEvent('locationfound', {
        lat: e.latlng.lat.toFixed(6),
        lng: e.latlng.lng.toFixed(6),
        accuracy: e.accuracy.toFixed(2),
      });
    },
    locationerror: (e) => logEvent('locationerror', { message: e.message }),

    // Other events
    resize: () => logEvent('resize'),
    load: () => logEvent('load'),
    unload: () => logEvent('unload'),
  });
```

```
    return null;
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Map Events</h3>
        <p>Interact with the map to see events logged here:</p>
        <div style={{ marginBottom: '10px' }}>
          <button onClick={() => setEventLog([])}>Clear Log</button>
          <button onClick={() => setClickedPoints([])}>Clear Points</button>
        </div>
      </div>

      <div style={{ display: 'flex', gap: '20px' }}>
        <MapContainer
          center={[51.505, -0.09]}
          zoom={13}
          style={{ height: '500px', width: '70%' }}
        >
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />

          <MapEventLogger />

          {/* Display markers at clicked points */}
          {clickedPoints.map((point, index) => (
            <Marker key={index} position={[point.lat, point.lng]}>
              <Popup>
                Clicked at {point.lat.toFixed(6)}, {point.lng.toFixed(6)}
              </Popup>
            </Marker>
          ))}
        </MapContainer>

        <div
          style={{
            width: '30%',
            height: '500px',
            overflowY: 'auto',
            border: '1px solid #ddd',
            padding: '10px',
          }}
        >
          <h4>Event Log</h4>
          {eventLog.length === 0 ? (
            <p>No events yet. Interact with the map.</p>
          ) : (
            <div>
              {eventLog.map((event, index) => (
```

```
                <div
                  key={index}
                  style={{
                    marginBottom: '8px',
                    padding: '8px',
                    borderBottom: '1px solid #eee',
                    fontSize: '0.9em',
                  }}
                >
                  <div>
                    <strong>{event.eventName}</strong> at {event.timestamp}
                  </div>
                  {Object.keys(event.data).length > 0 && (
                    <pre
                      style={{
                        margin: '5px 0 0 0',
                        padding: '5px',
                        backgroundColor: '#f5f5f5',
                        fontSize: '0.9em',
                        overflowX: 'auto',
                      }}
                    >
                      {JSON.stringify(event.data, null, 2)}
                    </pre>
                  )}
                </div>
              ))}
            </div>
          )}
        </div>
      </div>
    </div>
  );
};

export default BasicMapEventsExample;
```

Layer Events

In addition to map events, you can also listen to events on specific layers:

```
import React, { useState } from 'react';
import {
  MapContainer,
  TileLayer,
  Marker,
  Circle,
  Polygon,
  Tooltip,
} from 'react-leaflet';
```

```
const LayerEventsExample = () => {
  // State for active element and message
  const [activeElement, setActiveElement] = useState(null);
  const [interactionMessage, setInteractionMessage] = useState(
    'Interact with the elements on the map'
  );

  // Sample data
  const marker = {
    id: 'marker',
    position: [51.505, -0.09],
    name: 'Important Marker',
  };
  const circle = {
    id: 'circle',
    center: [51.51, -0.1],
    radius: 300,
    name: 'Interactive Circle',
  };
  const polygon = {
    id: 'polygon',
    positions: [
      [51.49, -0.08],
      [51.5, -0.06],
      [51.51, -0.07],
      [51.51, -0.09],
    ],
    name: 'Dynamic Polygon',
  };

  // Event handlers
  const handleMouseOver = (element) => {
    setActiveElement(element.id);
    setInteractionMessage(`Hovering over: ${element.name}`);
  };

  const handleMouseOut = () => {
    setActiveElement(null);
    setInteractionMessage('Interact with the elements on the map');
  };

  const handleClick = (element) => {
    setInteractionMessage(`Clicked on: ${element.name}`);
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Layer Events</h3>
        <p>
          <strong>Status:</strong> {interactionMessage}
        </p>
      </div>
```

```
        <MapContainer
          center={[51.505, -0.09]}
          zoom={13}
          style={{ height: '500px', width: '100%' }}
        >
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
  href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />

          {/* Interactive Marker */}
          <Marker
            position={marker.position}
            eventHandlers={{
              mouseover: () => handleMouseOver(marker),
              mouseout: handleMouseOut,
              click: () => handleClick(marker),
            }}
          >
            <Tooltip>
              {marker.name}
              <br />
              {activeElement === marker.id ? 'Active' : 'Hover to activate'}
            </Tooltip>
          </Marker>

          {/* Interactive Circle */}
          <Circle
            center={circle.center}
            radius={circle.radius}
            pathOptions={{
              color: activeElement === circle.id ? '#f44336' : '#2196f3',
              fillOpacity: activeElement === circle.id ? 0.7 : 0.4,
              weight: activeElement === circle.id ? 3 : 2,
            }}
            eventHandlers={{
              mouseover: () => handleMouseOver(circle),
              mouseout: handleMouseOut,
              click: () => handleClick(circle),
            }}
          >
            <Tooltip>
              {circle.name}
              <br />
              {activeElement === circle.id ? 'Active' : 'Hover to activate'}
            </Tooltip>
          </Circle>

          {/* Interactive Polygon */}
          <Polygon
            positions={polygon.positions}
            pathOptions={{
              color: activeElement === polygon.id ? '#9c27b0' : '#4caf50',
```

```
            fillOpacity: activeElement === polygon.id ? 0.7 : 0.4,
            weight: activeElement === polygon.id ? 3 : 2,
          }}
          eventHandlers={{
            mouseover: () => handleMouseOver(polygon),
            mouseout: handleMouseOut,
            click: () => handleClick(polygon),
          }}
        >
          <Tooltip>
            {polygon.name}
            <br />
            {activeElement === polygon.id ? 'Active' : 'Hover to activate'}
          </Tooltip>
        </Polygon>
      </MapContainer>
    </div>
  );
};

export default LayerEventsExample;
```

Keyboard Events and Accessibility

Making your map accessible also means handling keyboard events:

```
import React, { useEffect, useState } from 'react';
import { MapContainer, TileLayer, useMap, Marker, Popup } from 'react-leaflet';

// Component to handle keyboard controls
const KeyboardControls = ({ onKeyAction }) => {
  const map = useMap();

  useEffect(() => {
    const handleKeyDown = (e) => {
      // Only handle keys when map is focused
      if (!map.getContainer().contains(document.activeElement)) {
        return;
      }

      const key = e.key.toLowerCase();
      const panAmount = 100; // pixels

      switch (key) {
        case 'arrowup':
        case 'w':
          map.panBy([0, -panAmount]);
          onKeyAction('Pan up');
          break;
        case 'arrowdown':
        case 's':
```

```
            map.panBy([0, panAmount]);
            onKeyAction('Pan down');
            break;
          case 'arrowleft':
          case 'a':
            map.panBy([-panAmount, 0]);
            onKeyAction('Pan left');
            break;
          case 'arrowright':
          case 'd':
            map.panBy([panAmount, 0]);
            onKeyAction('Pan right');
            break;
          case '+':
          case '=':
            map.zoomIn();
            onKeyAction('Zoom in');
            break;
          case '-':
          case '_':
            map.zoomOut();
            onKeyAction('Zoom out');
            break;
          case 'r':
            map.setView([51.505, -0.09], 13);
            onKeyAction('Reset view');
            break;
          case 'f':
            if (e.ctrlKey || e.metaKey) {
              e.preventDefault(); // Prevent browser find
              map.locate({ setView: true, maxZoom: 16 });
              onKeyAction('Find my location');
            }
            break;
          default:
            // Other keys
            break;
        }
      };

      // Add event listener
      window.addEventListener('keydown', handleKeyDown);

      // Clean up
      return () => {
        window.removeEventListener('keydown', handleKeyDown);
      };
    }, [map, onKeyAction]);

    return null;
  };

  const KeyboardEventsExample = () => {
    const [lastAction, setLastAction] = useState('No actions yet');
```

```
    const [isFocused, setIsFocused] = useState(false);

    // Predefined locations
    const locations = [
      {
        id: 1,
        position: [51.505, -0.09],
        name: 'London',
        description: 'Capital of England',
      },
      {
        id: 2,
        position: [48.856, 2.352],
        name: 'Paris',
        description: 'Capital of France',
      },
      {
        id: 3,
        position: [41.902, 12.496],
        name: 'Rome',
        description: 'Capital of Italy',
      },
    ];

    const handleKeyAction = (action) => {
      setLastAction(action);
    };

    return (
      <div>
        <div style={{ marginBottom: '10px' }}>
          <h3>Keyboard Controls</h3>
          <p>
            <strong>Last Action:</strong> {lastAction}
          </p>
          <p>
            <strong>Status:</strong> Map is{' '}
            {isFocused ? 'focused' : 'not focused'}
          </p>

          <div style={{ marginBottom: '10px' }}>
            <strong>Keyboard Controls:</strong>
            <ul>
              <li>Arrow keys or WASD: Pan the map</li>
              <li>+ / -: Zoom in/out</li>
              <li>R: Reset view</li>
              <li>Ctrl+F: Find my location</li>
            </ul>
          </div>

          <div style={{ marginBottom: '10px' }}>
            <strong>Jump to location:</strong>
            {locations.map((location) => (
              <button
```

```
                key={location.id}
                style={{ marginRight: '10px' }}
                onClick={() => setLastAction(`Jumped to ${location.name}`)}
              >
                {location.name}
              </button>
            ))}
          </div>
        </div>

        <MapContainer
          center={[51.505, -0.09]}
          zoom={13}
          style={{ height: '500px', width: '100%' }}
          whenCreated={(map) => {
            // Make the map container accessible via keyboard
            const container = map.getContainer();
            container.tabIndex = 0;
            container.setAttribute(
              'aria-label',
              'Interactive map with keyboard controls'
            );

            container.addEventListener('focus', () => setIsFocused(true));
            container.addEventListener('blur', () => setIsFocused(false));
          }}
        >
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />

          <KeyboardControls onKeyAction={handleKeyAction} />

          {/* Markers for locations */}
          {locations.map((location) => (
            <Marker key={location.id} position={location.position}>
              <Popup>
                <div>
                  <h3>{location.name}</h3>
                  <p>{location.description}</p>
                </div>
              </Popup>
            </Marker>
          ))}
        </MapContainer>

        <div style={{ marginTop: '10px' }}>
          <button
            onClick={() => {
              // Focus the map
              const mapContainer = document.querySelector('.leaflet-container');
              if (mapContainer) {
```

```
              mapContainer.focus();
            }
          }}
        >
          Focus Map (for keyboard controls)
        </button>
      </div>
    </div>
  );
};

export default KeyboardEventsExample;
```

Creating Interactive Layers

Now let's create more sophisticated interactive layers that respond to user interactions:

Interactive Choropleth Map

A choropleth map shows different areas colored based on data values:

```
import React, { useState } from 'react';
import {
  MapContainer,
  TileLayer,
  GeoJSON,
  Tooltip,
  useMap,
} from 'react-leaflet';

// You would normally load this from an external file
const geoJsonData = {
  type: 'FeatureCollection',
  features: [
    {
      type: 'Feature',
      properties: {
        name: 'District A',
        value: 75,
        population: 250000,
        area: 120,
      },
      geometry: {
        type: 'Polygon',
        coordinates: [
          [
            [-0.12, 51.52],
            [-0.1, 51.52],
            [-0.1, 51.5],
            [-0.12, 51.5],
            [-0.12, 51.52],
```

```
          ],
        ],
      },
    },
    {
      type: 'Feature',
      properties: {
        name: 'District B',
        value: 45,
        population: 180000,
        area: 90,
      },
      geometry: {
        type: 'Polygon',
        coordinates: [
          [
            [-0.1, 51.52],
            [-0.08, 51.52],
            [-0.08, 51.5],
            [-0.1, 51.5],
            [-0.1, 51.52],
          ],
        ],
      },
    },
    {
      type: 'Feature',
      properties: {
        name: 'District C',
        value: 90,
        population: 320000,
        area: 150,
      },
      geometry: {
        type: 'Polygon',
        coordinates: [
          [
            [-0.12, 51.5],
            [-0.1, 51.5],
            [-0.1, 51.48],
            [-0.12, 51.48],
            [-0.12, 51.5],
          ],
        ],
      },
    },
    {
      type: 'Feature',
      properties: {
        name: 'District D',
        value: 30,
        population: 150000,
        area: 100,
      },
```

```
        geometry: {
          type: 'Polygon',
          coordinates: [
            [
              [-0.1, 51.5],
              [-0.08, 51.5],
              [-0.08, 51.48],
              [-0.1, 51.48],
              [-0.1, 51.5],
            ],
          ],
        },
      },
    ],
  };

  // Legend component
  const Legend = () => {
    return (
      <div
        className="info legend"
        style={{
          padding: '6px 8px',
          background: 'white',
          background: 'rgba(255,255,255,0.8)',
          boxShadow: '0 0 15px rgba(0,0,0,0.2)',
          borderRadius: '5px',
          position: 'absolute',
          bottom: '20px',
          right: '10px',
          zIndex: 1000,
        }}
      >
        <div style={{ marginBottom: '5px' }}>
          <strong>Value</strong>
        </div>
        {[0, 25, 50, 75].map((limit, i) => (
          <div
            key={i}
            style={{ display: 'flex', alignItems: 'center', marginBottom: '3px' }}
          >
            <div
              style={{
                width: '18px',
                height: '18px',
                marginRight: '8px',
                backgroundColor: getColor(limit + 1),
                opacity: 0.7,
              }}
            ></div>
            <span>
              {limit}
              {i < 3 ? '–' + (limit + 25) : '+'}
            </span>
```

```
          </div>
        ))}
      </div>
    );
};

// Function to get color based on value
const getColor = (value) => {
  return value > 75
    ? '#800026'
    : value > 50
    ? '#BD0026'
    : value > 25
    ? '#E31A1C'
    : '#FC4E2A';
};

// Info panel component
const InfoPanel = ({ feature }) => {
  return (
    <div
      className="info"
      style={{
        padding: '6px 8px',
        background: 'white',
        background: 'rgba(255,255,255,0.8)',
        boxShadow: '0 0 15px rgba(0,0,0,0.2)',
        borderRadius: '5px',
        position: 'absolute',
        top: '10px',
        right: '10px',
        zIndex: 1000,
        minWidth: '200px',
      }}
    >
      <h4 style={{ margin: '0 0 5px', color: '#777' }}>District Info</h4>
      {feature ? (
        <div>
          <strong>{feature.properties.name}</strong>
          <br />
          Value: {feature.properties.value}
          <br />
          Population: {feature.properties.population.toLocaleString()}
          <br />
          Area: {feature.properties.area} km²
        </div>
      ) : (
        <p>Hover over a district</p>
      )}
    </div>
  );
};

const InteractiveChoroplethMap = () => {
```

```
  const [selectedFeature, setSelectedFeature] = useState(null);
  const [dataMetric, setDataMetric] = useState('value');

  // Style function for GeoJSON
  const style = (feature) => {
    const value = feature.properties[dataMetric];
    const normalizedValue =
      dataMetric === 'value'
        ? value
        : dataMetric === 'population'
        ? value / 10000
        : value;

    return {
      fillColor: getColor(normalizedValue),
      weight: 2,
      opacity: 1,
      color: 'white',
      dashArray: '3',
      fillOpacity: 0.7,
    };
  };

  // Handle hover events
  const onEachFeature = (feature, layer) => {
    layer.on({
      mouseover: () => setSelectedFeature(feature),
      mouseout: () => setSelectedFeature(null),
      click: (e) => {
        // Highlight the clicked feature by changing its style
        layer.setStyle({
          weight: 5,
          color: '#666',
          dashArray: '',
          fillOpacity: 0.7,
        });

        // Bring to front
        if (!L.Browser.ie && !L.Browser.opera && !L.Browser.edge) {
          layer.bringToFront();
        }
      },
    });
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Interactive Choropleth Map</h3>
        <div>
          <span>Show map based on: </span>
          <select
            value={dataMetric}
            onChange={(e) => setDataMetric(e.target.value)}
```

```
            style={{ marginLeft: '10px' }}
          >
            <option value="value">Value</option>
            <option value="population">Population</option>
            <option value="area">Area</option>
          </select>
        </div>
      </div>

      <div style={{ position: 'relative' }}>
        <MapContainer
          center={[51.505, -0.09]}
          zoom={13}
          style={{ height: '600px', width: '100%' }}
        >
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />

          <GeoJSON
            key={dataMetric} // Force re-render when metric changes
            data={geoJsonData}
            style={style}
            onEachFeature={onEachFeature}
          />

          <Legend />
          <InfoPanel feature={selectedFeature} />
        </MapContainer>
      </div>
    </div>
  );
};

export default InteractiveChoroplethMap;
```

Interactive Timeline Map

This example shows how to create a map that displays data across time:

```
import React, { useState, useEffect } from 'react';
import {
  MapContainer,
  TileLayer,
  CircleMarker,
  Tooltip,
  useMap,
} from 'react-leaflet';
```

```
// Sample time series data (in real app, you'd load this from an API)
const timeSeriesData = [
  {
    timestamp: '2023-01-01',
    points: [
      { id: 1, position: [51.505, -0.09], value: 35, name: 'Location A' },
      { id: 2, position: [51.51, -0.1], value: 20, name: 'Location B' },
      { id: 3, position: [51.49, -0.11], value: 15, name: 'Location C' },
    ],
  },
  {
    timestamp: '2023-02-01',
    points: [
      { id: 1, position: [51.505, -0.09], value: 40, name: 'Location A' },
      { id: 2, position: [51.51, -0.1], value: 25, name: 'Location B' },
      { id: 3, position: [51.49, -0.11], value: 30, name: 'Location C' },
    ],
  },
  {
    timestamp: '2023-03-01',
    points: [
      { id: 1, position: [51.505, -0.09], value: 30, name: 'Location A' },
      { id: 2, position: [51.51, -0.1], value: 45, name: 'Location B' },
      { id: 3, position: [51.49, -0.11], value: 35, name: 'Location C' },
    ],
  },
  {
    timestamp: '2023-04-01',
    points: [
      { id: 1, position: [51.505, -0.09], value: 55, name: 'Location A' },
      { id: 2, position: [51.51, -0.1], value: 40, name: 'Location B' },
      { id: 3, position: [51.49, -0.11], value: 25, name: 'Location C' },
    ],
  },
];

// Component to update map view based on time
const TimeController = ({ currentTime, data }) => {
  const map = useMap();

  // Adjust view if needed based on time period
  useEffect(() => {
    // Example: if we want to change the view for certain time periods
    // You could also fit bounds to the active points

    // For demonstration, let's slightly shift the center based on time index
    const timeIndex = data.findIndex((d) => d.timestamp === currentTime);
    if (timeIndex >= 0) {
      const offset = (timeIndex - 1) * 0.005;
      map.setView([51.505 + offset, -0.09 + offset], 13);
    }
  }, [currentTime, map, data]);

  return null;
```

```
};

const TimelineMap = () => {
  const [currentTimeIndex, setCurrentTimeIndex] = useState(0);
  const [currentTime, setCurrentTime] = useState(timeSeriesData[0].timestamp);
  const [isPlaying, setIsPlaying] = useState(false);
  const [playbackSpeed, setPlaybackSpeed] = useState(1000); // milliseconds

  // Find current data based on selected time
  const currentData =
    timeSeriesData.find((d) => d.timestamp === currentTime) ||
    timeSeriesData[0];

  // Calculate the radius based on value
  const getRadius = (value) => {
    return 5 + value / 10;
  };

  // Get color based on value
  const getColor = (value) => {
    if (value >= 50) return '#d32f2f'; // high - red
    if (value >= 30) return '#ff9800'; // medium - orange
    return '#4caf50'; // low - green
  };

  // Handle play/pause
  useEffect(() => {
    let interval;

    if (isPlaying) {
      interval = setInterval(() => {
        setCurrentTimeIndex((prev) => {
          const nextIndex = (prev + 1) % timeSeriesData.length;
          setCurrentTime(timeSeriesData[nextIndex].timestamp);
          return nextIndex;
        });
      }, playbackSpeed);
    }

    return () => {
      if (interval) clearInterval(interval);
    };
  }, [isPlaying, playbackSpeed]);

  // Handle manual time selection
  const handleTimeSelection = (event) => {
    const selectedTime = event.target.value;
    setCurrentTime(selectedTime);
    setCurrentTimeIndex(
      timeSeriesData.findIndex((d) => d.timestamp === selectedTime)
    );
  };

  return (
```

```
      <div>
        <div style={{ marginBottom: '10px' }}>
          <h3>Timeline Map</h3>
          <p>
            Showing data for: <strong>{currentTime}</strong>
          </p>

          <div
            style={{
              marginBottom: '10px',
              display: 'flex',
              alignItems: 'center',
            }}
          >
            <button
              onClick={() => setIsPlaying(!isPlaying)}
              style={{ marginRight: '10px' }}
            >
              {isPlaying ? 'Pause' : 'Play'}
            </button>

            <select
              value={currentTime}
              onChange={handleTimeSelection}
              style={{ marginRight: '20px' }}
            >
              {timeSeriesData.map((d) => (
                <option key={d.timestamp} value={d.timestamp}>
                  {d.timestamp}
                </option>
              ))}
            </select>

            <div>
              <label>Speed: </label>
              <select
                value={playbackSpeed}
                onChange={(e) => setPlaybackSpeed(Number(e.target.value))}
              >
                <option value={2000}>Slow</option>
                <option value={1000}>Normal</option>
                <option value={500}>Fast</option>
              </select>
            </div>
          </div>
        </div>

        <MapContainer
          center={[51.505, -0.09]}
          zoom={13}
          style={{ height: '500px', width: '100%' }}
        >
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
```

```
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        <TimeController currentTime={currentTime} data={timeSeriesData} />

        {/* Display current points */}
        {currentData.points.map((point) => (
          <CircleMarker
            key={point.id}
            center={point.position}
            radius={getRadius(point.value)}
            pathOptions={{
              fillColor: getColor(point.value),
              color: getColor(point.value),
              fillOpacity: 0.6,
              weight: 2,
            }}
          >
            <Tooltip>
              <div>
                <strong>{point.name}</strong>
                <br />
                Value: {point.value}
              </div>
            </Tooltip>
          </CircleMarker>
        ))}
      </MapContainer>

      <div style={{ marginTop: '10px' }}>
        <h4>Legend</h4>
        <div style={{ display: 'flex', gap: '20px' }}>
          <div style={{ display: 'flex', alignItems: 'center' }}>
            <div
              style={{
                backgroundColor: '#4caf50',
                width: '20px',
                height: '20px',
                marginRight: '5px',
                borderRadius: '50%',
              }}
            ></div>
            <span>Low (0-29)</span>
          </div>
          <div style={{ display: 'flex', alignItems: 'center' }}>
            <div
              style={{
                backgroundColor: '#ff9800',
                width: '20px',
                height: '20px',
                marginRight: '5px',
                borderRadius: '50%',
              }}
```

```
              ></div>
              <span>Medium (30-49)</span>
            </div>
            <div style={{ display: 'flex', alignItems: 'center' }}>
              <div
                style={{
                  backgroundColor: '#d32f2f',
                  width: '20px',
                  height: '20px',
                  marginRight: '5px',
                  borderRadius: '50%',
                }}
              ></div>
              <span>High (50+)</span>
            </div>
          </div>
          <p>Circle size also increases with value</p>
        </div>
      </div>
    );
  };

  export default TimelineMap;
```

Drawing and Editing Geometries

Leaflet can be extended to allow users to draw and edit shapes on the map. For this, we'll use the `react-leaflet-draw` library which wraps Leaflet.Draw.

Basic Drawing Controls

```
import React, { useState, useRef } from 'react';
import { MapContainer, TileLayer, FeatureGroup } from 'react-leaflet';
import { EditControl } from 'react-leaflet-draw';
import 'leaflet/dist/leaflet.css';
import 'leaflet-draw/dist/leaflet.draw.css';

const DrawingToolsExample = () => {
  const [drawnItems, setDrawnItems] = useState([]);
  const featureGroupRef = useRef(null);

  // Event handlers
  const handleCreated = (e) => {
    const { layerType, layer } = e;

    // Get the geometry in GeoJSON format
    const geoJSON = layer.toGeoJSON();

    // Add type information (helps when displaying the list)
    geoJSON.properties.layerType = layerType;
```

```
    // For markers, simple coordinates might be more useful
    if (layerType === 'marker') {
      geoJSON.properties.coords = [
        layer.getLatLng().lat,
        layer.getLatLng().lng,
      ];
    }

    // Add to state
    setDrawnItems([...drawnItems, geoJSON]);
  };

  const handleEdited = (e) => {
    const { layers } = e;

    // Update drawnItems with edited geometries
    const editedIds = new Set();

    layers.eachLayer((layer) => {
      const editedGeoJSON = layer.toGeoJSON();
      editedGeoJSON.properties.layerType = layer.options.layerType;
      editedIds.add(layer._leaflet_id);

      setDrawnItems((prev) =>
        prev.map((item) =>
          item._leaflet_id === layer._leaflet_id ? editedGeoJSON : item
        )
      );
    });
  };

  const handleDeleted = (e) => {
    const { layers } = e;

    // Remove deleted items
    layers.eachLayer((layer) => {
      setDrawnItems((prev) =>
        prev.filter((item) => item._leaflet_id !== layer._leaflet_id)
      );
    });
  };

  // Format the shape description
  const getShapeDescription = (item) => {
    const { layerType } = item.properties;

    switch (layerType) {
      case 'marker':
        const [lat, lng] = item.properties.coords;
        return `Marker at [${lat.toFixed(6)}, ${lng.toFixed(6)}]`;
      case 'circle':
        return `Circle with radius ${item.properties.radius || 'unknown'}m`;
      case 'rectangle':
        return 'Rectangle';
```

```
        case 'polygon':
          return `Polygon with ${item.geometry.coordinates[0].length} points`;
        case 'polyline':
          return `Line with ${item.geometry.coordinates.length} points`;
        default:
          return layerType || 'Unknown shape';
      }
    };

    return (
      <div>
        <div style={{ marginBottom: '10px' }}>
          <h3>Drawing Tools</h3>
          <p>
            Use the drawing tools on the right side of the map to create shapes.
          </p>
        </div>

        <div style={{ display: 'flex', gap: '20px' }}>
          <MapContainer
            center={[51.505, -0.09]}
            zoom={13}
            style={{ height: '500px', width: '70%' }}
          >
            <TileLayer
              url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
              attribution='&copy; <a
  href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
            />

            <FeatureGroup ref={featureGroupRef}>
              <EditControl
                position="topright"
                onCreated={handleCreated}
                onEdited={handleEdited}
                onDeleted={handleDeleted}
                draw={{
                  rectangle: true,
                  polygon: true,
                  circle: true,
                  circlemarker: false,
                  marker: true,
                  polyline: true,
                }}
              />
            </FeatureGroup>
          </MapContainer>

          <div
            style={{
              width: '30%',
              padding: '10px',
              border: '1px solid #ddd',
              overflowY: 'auto',
```

```
            height: '500px',
          }}
        >
          <h4>Drawn Items</h4>
          {drawnItems.length === 0 ? (
            <p>No items drawn yet. Use the tools to create shapes.</p>
          ) : (
            <ul style={{ listStyleType: 'none', padding: 0 }}>
              {drawnItems.map((item, index) => (
                <li
                  key={index}
                  style={{
                    padding: '8px',
                    borderBottom: '1px solid #eee',
                    marginBottom: '5px',
                  }}
                >
                  <strong>{getShapeDescription(item)}</strong>
                  <br />
                  <small>
                    {item.properties.layerType === 'circle'
                      ? `Center: [${item.geometry.coordinates[1].toFixed(
                          6
                        )}, ${item.geometry.coordinates[0].toFixed(6)}]`
                      : ''}
                  </small>
                </li>
              ))}
            </ul>
          )}
        </div>
      </div>
    </div>
  );
};

export default DrawingToolsExample;
```

Saving and Loading Drawn Shapes

```
import React, { useState, useRef, useEffect } from 'react';
import { MapContainer, TileLayer, FeatureGroup, GeoJSON } from 'react-leaflet';
import { EditControl } from 'react-leaflet-draw';
import L from 'leaflet';
import 'leaflet/dist/leaflet.css';
import 'leaflet-draw/dist/leaflet.draw.css';

const SaveLoadDrawingsExample = () => {
  const [drawnShapes, setDrawnShapes] = useState([]);
  const featureGroupRef = useRef(null);
```

```
  // On mount, try to load saved shapes from localStorage
  useEffect(() => {
    try {
      const saved = localStorage.getItem('leaflet-drawn-shapes');
      if (saved) {
        setDrawnShapes(JSON.parse(saved));
      }
    } catch (err) {
      console.error('Failed to load saved shapes', err);
    }
  }, []);

  // Save shapes to localStorage when they change
  useEffect(() => {
    if (drawnShapes.length > 0) {
      localStorage.setItem('leaflet-drawn-shapes', JSON.stringify(drawnShapes));
    }
  }, [drawnShapes]);

  // When shapes are loaded, add them to the FeatureGroup
  useEffect(() => {
    if (!featureGroupRef.current || drawnShapes.length === 0) return;

    // Clear existing layers
    featureGroupRef.current.clearLayers();

    // Create layers from GeoJSON
    const geoJSONLayer = L.geoJSON(
      {
        type: 'FeatureCollection',
        features: drawnShapes,
      },
      {
        style: (feature) => {
          // You can customize styles based on properties
          return feature.properties.style || {};
        },
        pointToLayer: (feature, latlng) => {
          if (feature.properties.layerType === 'circle') {
            return L.circle(latlng, {
              radius: feature.properties.radius || 100,
            });
          }
          return L.marker(latlng);
        },
      }
    );

    // Add all layers to the feature group
    geoJSONLayer.eachLayer((layer) => {
      featureGroupRef.current.addLayer(layer);
    });
  }, [drawnShapes]);
```

```
    // Event handlers
    const handleCreated = (e) => {
      const { layerType, layer } = e;

      // Get the geometry in GeoJSON format
      const newShape = layer.toGeoJSON();

      // Add layer type to properties
      newShape.properties.layerType = layerType;

      // Add radius for circles (not included in toGeoJSON)
      if (layerType === 'circle') {
        newShape.properties.radius = layer.getRadius();
      }

      // Store style information
      if (layer.options) {
        newShape.properties.style = {
          color: layer.options.color,
          weight: layer.options.weight,
          opacity: layer.options.opacity,
          fillColor: layer.options.fillColor,
          fillOpacity: layer.options.fillOpacity,
        };
      }

      // Set internal ID to help with updates
      newShape._id = `shape_${Date.now()}`;

      // Add to state
      setDrawnShapes([...drawnShapes, newShape]);
    };

    const handleEdited = (e) => {
      const { layers } = e;
      const editedShapes = [...drawnShapes];

      layers.eachLayer((layer) => {
        const editedShape = layer.toGeoJSON();

        // Copy over properties from the original shape
        const originalIndex = drawnShapes.findIndex(
          (s) => s._id === layer.feature?._id
        );

        if (originalIndex !== -1) {
          editedShape.properties = {
            ...drawnShapes[originalIndex].properties,
            ...editedShape.properties,
          };

          // Update radius for circles
          if (editedShape.properties.layerType === 'circle') {
            editedShape.properties.radius = layer.getRadius();
```

```
        }

        editedShape._id = drawnShapes[originalIndex]._id;
        editedShapes[originalIndex] = editedShape;
      }
    });

    setDrawnShapes(editedShapes);
  };

  const handleDeleted = (e) => {
    const { layers } = e;
    const deletedIds = new Set();

    layers.eachLayer((layer) => {
      if (layer.feature && layer.feature._id) {
        deletedIds.add(layer.feature._id);
      }
    });

    if (deletedIds.size > 0) {
      const remainingShapes = drawnShapes.filter(
        (shape) => !deletedIds.has(shape._id)
      );
      setDrawnShapes(remainingShapes);
    }
  };

  // Clear all shapes
  const handleClearAll = () => {
    if (featureGroupRef.current) {
      featureGroupRef.current.clearLayers();
    }
    setDrawnShapes([]);
    localStorage.removeItem('leaflet-drawn-shapes');
  };

  // Export shapes as GeoJSON
  const handleExport = () => {
    if (drawnShapes.length === 0) {
      alert('No shapes to export');
      return;
    }

    const geoJSON = {
      type: 'FeatureCollection',
      features: drawnShapes,
    };

    const dataStr = JSON.stringify(geoJSON, null, 2);
    const dataUri =
      'data:application/json;charset=utf-8,' + encodeURIComponent(dataStr);

    // Create a download link and trigger it
```

```
    const exportLink = document.createElement('a');
    exportLink.setAttribute('href', dataUri);
    exportLink.setAttribute('download', 'map_shapes.geojson');
    document.body.appendChild(exportLink);
    exportLink.click();
    document.body.removeChild(exportLink);
  };

  // Import GeoJSON
  const handleImport = (event) => {
    const file = event.target.files[0];
    if (!file) return;

    const reader = new FileReader();
    reader.onload = (e) => {
      try {
        const imported = JSON.parse(e.target.result);

        // Check if it's a valid GeoJSON
        if (imported.type === 'FeatureCollection') {
          // Assign IDs to imported features if they don't have one
          const featuresWithIds = imported.features.map((feature) => ({
            ...feature,
            _id:
              feature._id ||
              `shape_${Date.now()}_${Math.random().toString(36).substr(2, 9)}`,
          }));

          setDrawnShapes(featuresWithIds);
        } else {
          alert('Invalid GeoJSON format');
        }
      } catch (err) {
        console.error('Failed to parse imported file', err);
        alert('Failed to import: ' + err.message);
      }
    };
    reader.readAsText(file);

    // Reset the input so the same file can be selected again
    event.target.value = null;
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Save and Load Drawings</h3>
        <p>
          Draw shapes on the map. They will be automatically saved to your
          browser's local storage.
        </p>

        <div style={{ marginBottom: '10px', display: 'flex', gap: '10px' }}>
          <button onClick={handleExport} disabled={drawnShapes.length === 0}>
```

```
            Export to GeoJSON
          </button>

          <label
            style={{
              padding: '6px 12px',
              border: '1px solid #ccc',
              borderRadius: '4px',
              backgroundColor: '#f8f9fa',
              cursor: 'pointer',
            }}
          >
            Import GeoJSON
            <input
              type="file"
              accept=".json,.geojson"
              onChange={handleImport}
              style={{ display: 'none' }}
            />
          </label>

          <button
            onClick={handleClearAll}
            disabled={drawnShapes.length === 0}
            style={{ backgroundColor: '#f44336', color: 'white' }}
          >
            Clear All Shapes
          </button>
        </div>
      </div>

      <MapContainer
        center={[51.505, -0.09]}
        zoom={13}
        style={{ height: '500px', width: '100%' }}
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        <FeatureGroup ref={featureGroupRef}>
          <EditControl
            position="topright"
            onCreated={handleCreated}
            onEdited={handleEdited}
            onDeleted={handleDeleted}
            draw={{
              rectangle: true,
              polygon: {
                allowIntersection: false,
                drawError: {
                  color: '#e1e100',
```

```
                  message: '<strong>Error:</strong> Shape edges cannot cross!',
                },
                shapeOptions: {
                  color: '#97009c',
                },
              },
              circle: true,
              circlemarker: false,
              marker: true,
              polyline: {
                shapeOptions: {
                  color: '#f357a1',
                  weight: 3,
                },
              },
            }}
            edit={{
              featureGroup: featureGroupRef.current,
              remove: true,
              edit: true,
            }}
          />
        </FeatureGroup>
      </MapContainer>

      <div style={{ marginTop: '10px' }}>
        <h4>Shape Statistics</h4>
        <p>
          Total shapes: <strong>{drawnShapes.length}</strong>
          <br />
          {drawnShapes.length > 0 && (
            <>
              Types:{' '}
              {Object.entries(
                drawnShapes.reduce((acc, shape) => {
                  const type = shape.properties.layerType;
                  acc[type] = (acc[type] || 0) + 1;
                  return acc;
                }, {})
              )
                .map(([type, count]) => `${type} (${count})`)
                .join(', ')}
            </>
          )}
        </p>
      </div>
    </div>
  );
};

export default SaveLoadDrawingsExample;
```

Measuring Distances and Areas

Let's implement tools for measuring distances and areas on the map:

```
import React, { useState, useEffect } from 'react';
import {
  MapContainer,
  TileLayer,
  Polyline,
  Polygon,
  useMapEvents,
} from 'react-leaflet';
import L from 'leaflet';

// Helper function to calculate distance between two points
const calculateDistance = (point1, point2) => {
  return L.latLng(point1).distanceTo(L.latLng(point2));
};

// Helper function to calculate area of a polygon
const calculateArea = (points) => {
  if (points.length < 3) return 0;

  // Create a Leaflet polygon to calculate area
  const latLngs = points.map((point) => L.latLng(point));
  const polygon = L.polygon(latLngs);

  return L.GeometryUtil.geodesicArea(polygon.getLatLngs()[0]);
};

const MeasuringTools = () => {
  const [measureMode, setMeasureMode] = useState(null); // 'distance' or 'area'
  const [measurePoints, setMeasurePoints] = useState([]);
  const [distance, setDistance] = useState(0);
  const [area, setArea] = useState(0);

  // Format distance nicely
  const formatDistance = (meters) => {
    if (meters < 1000) {
      return `${meters.toFixed(1)} m`;
    } else {
      return `${(meters / 1000).toFixed(2)} km`;
    }
  };

  // Format area nicely
  const formatArea = (squareMeters) => {
    if (squareMeters < 10000) {
      return `${squareMeters.toFixed(1)} m²`;
    } else {
      return `${(squareMeters / 10000).toFixed(2)} ha`;
    }
```

```
  };

  // Reset the measuring tool
  const resetMeasure = () => {
    setMeasurePoints([]);
    setDistance(0);
    setArea(0);
  };

  // Update measurements when points change
  useEffect(() => {
    if (measureMode === 'distance' && measurePoints.length >= 2) {
      // Calculate total distance along the path
      let totalDistance = 0;
      for (let i = 1; i < measurePoints.length; i++) {
        totalDistance += calculateDistance(
          measurePoints[i - 1],
          measurePoints[i]
        );
      }
      setDistance(totalDistance);
    } else if (measureMode === 'area' && measurePoints.length >= 3) {
      // Calculate polygon area
      setArea(calculateArea(measurePoints));
    }
  }, [measurePoints, measureMode]);

  // Map click handler component
  const MeasureEventHandler = () => {
    useMapEvents({
      click: (e) => {
        if (measureMode) {
          // Add point to measurement
          setMeasurePoints((prev) => [...prev, [e.latlng.lat, e.latlng.lng]]);
        }
      },
    });
    return null;
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Measuring Tools</h3>
        <p>Select a tool and click on the map to start measuring:</p>

        <div style={{ marginBottom: '10px', display: 'flex', gap: '10px' }}>
          <button
            onClick={() => {
              resetMeasure();
              setMeasureMode('distance');
            }}
            style={{
              backgroundColor:
```

```
                measureMode === 'distance' ? '#2196f3' : '#f8f9fa',
              color: measureMode === 'distance' ? 'white' : 'black',
            }}
          >
            Measure Distance
          </button>

          <button
            onClick={() => {
              resetMeasure();
              setMeasureMode('area');
            }}
            style={{
              backgroundColor: measureMode === 'area' ? '#4caf50' : '#f8f9fa',
              color: measureMode === 'area' ? 'white' : 'black',
            }}
          >
            Measure Area
          </button>

          <button onClick={resetMeasure}>Clear Measurements</button>
        </div>

        {/* Display measurement results */}
        {measureMode === 'distance' && measurePoints.length > 0 && (
          <div>
            <p>
              <strong>Distance:</strong> {formatDistance(distance)}
              <br />
              <strong>Points:</strong> {measurePoints.length}
            </p>
          </div>
        )}

        {measureMode === 'area' && measurePoints.length > 0 && (
          <div>
            <p>
              <strong>Area:</strong> {formatArea(area)}
              <br />
              <strong>Perimeter:</strong> {formatDistance(distance)}
              <br />
              <strong>Points:</strong> {measurePoints.length}
            </p>
          </div>
        )}
      </div>

      <MapContainer
        center={[51.505, -0.09]}
        zoom={13}
        style={{ height: '500px', width: '100%' }}
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
```

```
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        <MeasureEventHandler />

        {/* Draw the measurement line or polygon */}
        {measureMode === 'distance' && measurePoints.length > 1 && (
          <Polyline
            positions={measurePoints}
            pathOptions={{ color: '#2196f3', weight: 4 }}
          />
        )}

        {measureMode === 'area' && measurePoints.length > 2 && (
          <Polygon
            positions={measurePoints}
            pathOptions={{
              color: '#4caf50',
              weight: 3,
              fillOpacity: 0.2,
            }}
          />
        )}
      </MapContainer>

      <div style={{ marginTop: '10px' }}>
        <h4>Instructions</h4>
        <ul>
          <li>Select "Measure Distance" to calculate the length of a path.</li>
          <li>Select "Measure Area" to calculate the area of a polygon.</li>
          <li>Click on the map to add measurement points.</li>
          <li>Click "Clear Measurements" to start over.</li>
        </ul>
      </div>
    </div>
  );
};

export default MeasuringTools;
```

User Location and Tracking

Now let's implement features to locate and track the user's position:

```
import React, { useState, useEffect, useCallback } from 'react';
import {
  MapContainer,
  TileLayer,
  Marker,
  Circle,
```

```
  Polyline,
  useMap,
} from 'react-leaflet';
import L from 'leaflet';

// Component to handle location updates and map view
const LocationHandler = ({
  setPosition,
  setAccuracy,
  setLocationHistory,
  followUser,
}) => {
  const map = useMap();

  // Function to handle successful location
  const handleLocationFound = useCallback(
    (e) => {
      const { lat, lng } = e.latlng;
      const accuracy = e.accuracy;

      // Update current position
      setPosition([lat, lng]);
      setAccuracy(accuracy);

      // Add to location history with timestamp
      setLocationHistory((prev) => [
        ...prev,
        { position: [lat, lng], timestamp: new Date().toISOString() },
      ]);

      // Center map on user if followUser is enabled
      if (followUser) {
        map.setView([lat, lng]);
      }
    },
    [map, setPosition, setAccuracy, setLocationHistory, followUser]
  );

  // Function to handle location errors
  const handleLocationError = useCallback((e) => {
    console.error('Location error:', e.message);
    alert(`Error getting location: ${e.message}`);
  }, []);

  // Set up location watching when component mounts
  useEffect(() => {
    map.on('locationfound', handleLocationFound);
    map.on('locationerror', handleLocationError);

    // Start locating
    map.locate({ setView: true, maxZoom: 16 });

    // Set up continuous tracking if supported
    let locateInterval;
```

```
    if (navigator.geolocation) {
      locateInterval = setInterval(() => {
        map.locate({ setView: followUser });
      }, 5000); // Update every 5 seconds
    }

    return () => {
      map.off('locationfound', handleLocationFound);
      map.off('locationerror', handleLocationError);
      clearInterval(locateInterval);
    };
  }, [map, handleLocationFound, handleLocationError, followUser]);

  return null;
};

// Custom location marker icon
const createLocationIcon = () => {
  return L.divIcon({
    className: 'custom-location-icon',
    html: `
      <div style="
        background-color: #3388ff;
        border: 2px solid white;
        border-radius: 50%;
        width: 16px;
        height: 16px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);
      "></div>
    `,
    iconSize: [16, 16],
    iconAnchor: [8, 8],
  });
};

const UserLocationExample = () => {
  const [position, setPosition] = useState(null);
  const [accuracy, setAccuracy] = useState(0);
  const [followUser, setFollowUser] = useState(true);
  const [locationHistory, setLocationHistory] = useState([]);
  const [showHistory, setShowHistory] = useState(true);

  // Calculate path distance
  const calculatePathDistance = () => {
    if (locationHistory.length < 2) return 0;

    let totalDistance = 0;
    for (let i = 1; i < locationHistory.length; i++) {
      const point1 = locationHistory[i - 1].position;
      const point2 = locationHistory[i].position;
      totalDistance += L.latLng(point1).distanceTo(L.latLng(point2));
    }

    return totalDistance;
```

```
  };

  // Format a distance nicely
  const formatDistance = (meters) => {
    if (meters < 1000) {
      return `${meters.toFixed(1)} m`;
    } else {
      return `${(meters / 1000).toFixed(2)} km`;
    }
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>User Location Tracking</h3>

        {position ? (
          <div>
            <p>
              <strong>Current Position:</strong> {position[0].toFixed(6)},{' '}
              {position[1].toFixed(6)}
              <br />
              <strong>Accuracy:</strong> {accuracy.toFixed(1)} meters
              <br />
              <strong>Points Recorded:</strong> {locationHistory.length}
              <br />
              {locationHistory.length > 1 && (
                <>
                  <strong>Distance Traveled:</strong>{' '}
                  {formatDistance(calculatePathDistance())}
                </>
              )}
            </p>
          </div>
        ) : (
          <p>Waiting for location...</p>
        )}

        <div style={{ display: 'flex', gap: '10px', marginBottom: '10px' }}>
          <label>
            <input
              type="checkbox"
              checked={followUser}
              onChange={(e) => setFollowUser(e.target.checked)}
            />
            Follow my location
          </label>

          <label>
            <input
              type="checkbox"
              checked={showHistory}
              onChange={(e) => setShowHistory(e.target.checked)}
            />
```

```
            Show location history
          </label>

          <button onClick={() => setLocationHistory([])}>Clear History</button>
        </div>
      </div>

      <MapContainer
        center={[51.505, -0.09]}
        zoom={13}
        style={{ height: '500px', width: '100%' }}
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        <LocationHandler
          setPosition={setPosition}
          setAccuracy={setAccuracy}
          setLocationHistory={setLocationHistory}
          followUser={followUser}
        />

        {/* Display current location */}
        {position && (
          <>
            <Marker position={position} icon={createLocationIcon()} />

            <Circle
              center={position}
              radius={accuracy}
              pathOptions={{
                color: '#3388ff',
                weight: 1,
                fillColor: '#3388ff',
                fillOpacity: 0.15,
              }}
            />
          </>
        )}

        {/* Display location history */}
        {showHistory && locationHistory.length > 1 && (
          <Polyline
            positions={locationHistory.map((item) => item.position)}
            pathOptions={{
              color: '#ff3333',
              weight: 3,
              dashArray: '5, 5',
            }}
          />
        )}
```

```
      </MapContainer>

      <div style={{ marginTop: '10px' }}>
        <h4>Notes on Location Tracking</h4>
        <ul>
          <li>
            Location accuracy depends on your device's GPS capabilities and
            environment.
          </li>
          <li>
            In production, consider throttling location updates to save battery.
          </li>
          <li>The browser will ask for permission to access your location.</li>
          <li>
            For testing on desktop, browsers may use IP-based location which is
            less accurate.
          </li>
        </ul>
      </div>
    </div>
  );
};

export default UserLocationExample;
```

Advanced Features

Custom Projections

Leaflet uses Web Mercator (EPSG:3857) by default, but you can use other projections for specialized maps:

```
import React from 'react';
import { MapContainer, TileLayer, Rectangle, Tooltip } from 'react-leaflet';
import L from 'leaflet';
import 'proj4leaflet'; // Import the proj4leaflet library

// Define a custom projection
const setupCustomProjection = () => {
  // Example: UTM Zone 11N (for Western US)
  const utm11nProj = '+proj=utm +zone=11 +datum=WGS84 +units=m +no_defs';

  const customCRS = new L.Proj.CRS(
    'EPSG:32611', // UTM Zone 11N
    utm11nProj,
    {
      resolutions: [
        8192, 4096, 2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1, 0.5
      ],
      origin: [0, 0]
    }
  );
```

```
  return customCRS;
};

// Component with standard Web Mercator projection (default)
const StandardProjectionMap = () => {
  return (
    <div>
      <h4>Web Mercator Projection (EPSG:3857)</h4>
      <MapContainer
        center={[0, 0]}
        zoom={1}
        style={{ height: '400px', width: '100%' }}
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        {/* Add some rectangles to highlight distortion */}
        <Rectangle
          bounds={[[-80, -150], [-70, -130]]}
          pathOptions={{ color: 'red' }}
        >
          <Tooltip direction="center" permanent>
            Equal area at -75° latitude
          </Tooltip>
        </Rectangle>

        <Rectangle
          bounds={[[70, -150], [80, -130]]]
          pathOptions={{ color: 'red' }}
        >
          <Tooltip direction="center" permanent>
            Equal area at +75° latitude
          </Tooltip>
        </Rectangle>

        <Rectangle
          bounds={[[-5, -150], [5, -130]]}
          pathOptions={{ color: 'green' }}
        >
          <Tooltip direction="center" permanent>
            Equal area at equator
          </Tooltip>
        </Rectangle>
      </MapContainer>
    </div>
  );
};

// Component with a custom projection
const CustomProjectionMap = () => {
```

```
  const customCRS = setupCustomProjection();

  return (
    <div>
      <h4>UTM Zone 11N Projection (EPSG:32611)</h4>
      <div style={{ color: 'red' }}>
        <strong>Note:</strong> This is for illustration only. In a real app, you
would use tiles generated specifically for this projection.
      </div>
      <MapContainer
        center={[36.7783, -119.4179]} // California
        zoom={5}
        style={{ height: '400px', width: '100%' }}
        crs={customCRS}
      >
        {/* For a real app, you'd use tiles in the correct projection */}
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />
      </MapContainer>
    </div>
  );
};

const ProjectionsExample = () => {
  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Custom Projections</h3>
        <p>Maps can use different projections depending on your needs:</p>
        <ul>
          <li><strong>Web Mercator (EPSG:3857)</strong>: Standard for web maps,
preserves shape but distorts size at high latitudes</li>
          <li><strong>Equirectangular (EPSG:4326)</strong>: Simple lat/lng grid,
good for thematic maps</li>
          <li><strong>UTM</strong>: Great for local accuracy in specific
regions</li>
          <li><strong>Equal Area</strong>: Preserves area, good for data
visualization</li>
        </ul>
      </div>

      <StandardProjectionMap />

      <div style={{ height: '20px' }}></div>

      <CustomProjectionMap />

      <div style={{ marginTop: '20px' }}>
        <h4>When to Use Custom Projections</h4>
        <ul>
          <li>For maps covering polar regions (where Mercator distortion is
```

```
extreme)</li>
          <li>For thematic maps where accurate area representation is
important</li>
          <li>For specialized applications like surveying or engineering</li>
          <li>When working with local coordinate systems or data in specific
projections</li>
        </ul>

        <h4>Implementation Notes</h4>
        <ul>
          <li>Custom projections require additional libraries like
<code>proj4</code> and <code>proj4leaflet</code></li>
          <li>Tile services are typically designed for Web Mercator - custom
projections may need custom tile sources</li>
          <li>Converting coordinates between projections may introduce small
inaccuracies</li>
        </ul>
      </div>
    </div>
  );
};

export default ProjectionsExample;
```

Marker Clustering

When you have hundreds or thousands of markers, clustering them improves performance and readability:

```
import React, { useState, useEffect } from 'react';
import { MapContainer, TileLayer, Marker, Popup } from 'react-leaflet';
import MarkerClusterGroup from 'react-leaflet-cluster';
import L from 'leaflet';
import 'leaflet/dist/leaflet.css';
import 'react-leaflet-cluster/lib/assets/MarkerCluster.css';
import 'react-leaflet-cluster/lib/assets/MarkerCluster.Default.css';

const MarkerClusteringExample = () => {
  const [points, setPoints] = useState([]);
  const [clusteringEnabled, setClusteringEnabled] = useState(true);
  const [loading, setLoading] = useState(true);

  // Generate random points for demonstration
  useEffect(() => {
    const generateRandomPoints = (count) => {
      // Generate points around London
      const basePosition = [51.505, -0.09];
      const points = [];

      for (let i = 0; i < count; i++) {
        // Random offset roughly within ~5km
        const latOffset = (Math.random() - 0.5) * 0.1;
```

```
        const lngOffset = (Math.random() - 0.5) * 0.1;

        points.push({
          id: i,
          position: [basePosition[0] + latOffset, basePosition[1] + lngOffset],
          name: `Point ${i + 1}`,
          category: ['A', 'B', 'C'][Math.floor(Math.random() * 3)],
        });
      }

      return points;
    };

    // Generate 500 random points
    setLoading(true);
    setPoints(generateRandomPoints(500));
    setLoading(false);
  }, []);

  // Create category-specific icons
  const categoryIcons = {
    A: L.divIcon({
      html: `<div style="
        background-color: #e53935;
        width: 10px;
        height: 10px;
        border-radius: 50%;
        border: 2px solid white;
      "></div>`,
      className: 'custom-div-icon',
      iconSize: [15, 15],
      iconAnchor: [7, 7],
    }),
    B: L.divIcon({
      html: `<div style="
        background-color: #43a047;
        width: 10px;
        height: 10px;
        border-radius: 50%;
        border: 2px solid white;
      "></div>`,
      className: 'custom-div-icon',
      iconSize: [15, 15],
      iconAnchor: [7, 7],
    }),
    C: L.divIcon({
      html: `<div style="
        background-color: #1e88e5;
        width: 10px;
        height: 10px;
        border-radius: 50%;
        border: 2px solid white;
      "></div>`,
      className: 'custom-div-icon',
```

```
        iconSize: [15, 15],
        iconAnchor: [7, 7],
      }),
    };

    // Custom cluster icon creator function
    const createClusterCustomIcon = (cluster) => {
      const count = cluster.getChildCount();
      let size;

      // Adjust size based on number of points
      if (count < 10) {
        size = 30;
      } else if (count < 100) {
        size = 40;
      } else {
        size = 50;
      }

      // Category counts for pie chart (in real app, you'd get this from actual
data)
      const categories = {
        A: Math.round(count * 0.4), // 40% category A
        B: Math.round(count * 0.35), // 35% category B
        C: Math.round(count * 0.25), // 25% category C
      };

      // Custom HTML for cluster icon (simplified representation)
      return L.divIcon({
        html: `
          <div style="
            background-color: rgba(255,255,255,0.8);
            border-radius: 50%;
            width: ${size}px;
            height: ${size}px;
            display: flex;
            justify-content: center;
            align-items: center;
            font-weight: bold;
            color: #333;
            box-shadow: 0 0 5px rgba(0,0,0,0.3);
            border: 3px solid #fff;
            font-size: ${Math.max(12, Math.min(size * 0.4, 16))}px;
          ">
            ${count}
          </div>
        `,
        className: 'custom-cluster-icon',
        iconSize: [size, size],
        iconAnchor: [size / 2, size / 2],
      });
    };

    return (
```

```
      <div>
        <div style={{ marginBottom: '10px' }}>
          <h3>Marker Clustering</h3>
          {loading ? (
            <p>Loading points...</p>
          ) : (
            <p>Showing {points.length} points on the map.</p>
          )}

          <div style={{ marginBottom: '10px' }}>
            <label>
              <input
                type="checkbox"
                checked={clusteringEnabled}
                onChange={(e) => setClusteringEnabled(e.target.checked)}
              />
              Enable clustering
            </label>
          </div>

          <div style={{ marginBottom: '10px' }}>
            <strong>Categories:</strong>
            <div style={{ display: 'flex', gap: '15px', marginTop: '5px' }}>
              <div style={{ display: 'flex', alignItems: 'center' }}>
                <div
                  style={{
                    backgroundColor: '#e53935',
                    width: '12px',
                    height: '12px',
                    borderRadius: '50%',
                    marginRight: '5px',
                  }}
                ></div>
                <span>Category A</span>
              </div>
              <div style={{ display: 'flex', alignItems: 'center' }}>
                <div
                  style={{
                    backgroundColor: '#43a047',
                    width: '12px',
                    height: '12px',
                    borderRadius: '50%',
                    marginRight: '5px',
                  }}
                ></div>
                <span>Category B</span>
              </div>
              <div style={{ display: 'flex', alignItems: 'center' }}>
                <div
                  style={{
                    backgroundColor: '#1e88e5',
                    width: '12px',
                    height: '12px',
                    borderRadius: '50%',
```

```
                    marginRight: '5px',
                  }}
                ></div>
                <span>Category C</span>
              </div>
            </div>
          </div>
        </div>

        <MapContainer
          center={[51.505, -0.09]}
          zoom={13}
          style={{ height: '500px', width: '100%' }}
        >
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />

          {!loading &&
            (clusteringEnabled ? (
              <MarkerClusterGroup
                chunkedLoading
                iconCreateFunction={createClusterCustomIcon}
                maxClusterRadius={60}
                spiderfyOnMaxZoom={true}
                showCoverageOnHover={true}
                zoomToBoundsOnClick={true}
              >
                {points.map((point) => (
                  <Marker
                    key={point.id}
                    position={point.position}
                    icon={categoryIcons[point.category]}
                  >
                    <Popup>
                      <div>
                        <strong>{point.name}</strong>
                        <br />
                        Category: {point.category}
                      </div>
                    </Popup>
                  </Marker>
                ))}
              </MarkerClusterGroup>
            ) : (
              // Render markers without clustering
              points.map((point) => (
                <Marker
                  key={point.id}
                  position={point.position}
                  icon={categoryIcons[point.category]}
                >
```

```
                <Popup>
                  <div>
                    <strong>{point.name}</strong>
                    <br />
                    Category: {point.category}
                  </div>
                </Popup>
              </Marker>
            ))
          ))}
        </MapContainer>

        <div style={{ marginTop: '15px' }}>
          <h4>Marker Clustering Benefits</h4>
          <ul>
            <li>
              <strong>Performance</strong>: Significantly improves render times
              and reduces DOM elements
            </li>
            <li>
              <strong>Usability</strong>: Prevents overlapping markers and makes
              the map more readable
            </li>
            <li>
              <strong>Visual Appeal</strong>: Provides a cleaner interface that
              progressively reveals detail
            </li>
          </ul>

          <h4>Implementation Notes</h4>
          <ul>
            <li>
              The <code>react-leaflet-cluster</code> package provides integration
              with React Leaflet
            </li>
            <li>
              Key parameters to customize: <code>maxClusterRadius</code>,{' '}
              <code>spiderfyOnMaxZoom</code>, <code>showCoverageOnHover</code>
            </li>
            <li>
              You can create custom cluster icons that represent the data within
              the cluster
            </li>
            <li>
              For very large datasets (10,000+ points), consider server-side
              clustering or data aggregation
            </li>
          </ul>
        </div>
      </div>
    );
  };

  export default MarkerClusteringExample;
```

Heatmaps and Data Visualization

Heatmaps are excellent for visualizing density and distribution of point data:

```
import React, { useState, useEffect } from 'react';
import { MapContainer, TileLayer, useMap } from 'react-leaflet';
import L from 'leaflet';
import 'leaflet.heat'; // Import the Leaflet.heat plugin
import 'leaflet/dist/leaflet.css';

// Component to handle the heatmap layer
const HeatmapLayer = ({ points, intensity, radius, blur }) => {
  const map = useMap();

  useEffect(() => {
    if (!map || !points.length) return;

    // Create and add the heat layer
    const heatLayer = L.heatLayer(points, {
      radius: radius,
      blur: blur,
      maxZoom: 17,
      max: 1.0,
      gradient: {
        0.2: 'blue',
        0.4: 'lime',
        0.6: 'yellow',
        0.8: 'orange',
        1.0: 'red',
      },
    }).addTo(map);

    // Clean up on unmount
    return () => {
      map.removeLayer(heatLayer);
    };
  }, [map, points, intensity, radius, blur]);

  return null;
};

const HeatmapExample = () => {
  const [heatmapPoints, setHeatmapPoints] = useState([]);
  const [heatmapRadius, setHeatmapRadius] = useState(25);
  const [heatmapBlur, setHeatmapBlur] = useState(15);
  const [heatmapIntensity, setHeatmapIntensity] = useState(0.5);
  const [dataSet, setDataSet] = useState('uniform');
  const [loading, setLoading] = useState(true);

  // Generate heatmap data
  useEffect(() => {
```

```
    setLoading(true);

    // Generate different distributions of points
    const generateHeatmapData = (type, count = 1000) => {
      const points = [];

      if (type === 'uniform') {
        // Uniform distribution around London
        for (let i = 0; i < count; i++) {
          const lat = 51.505 + (Math.random() - 0.5) * 0.1;
          const lng = -0.09 + (Math.random() - 0.5) * 0.1;
          // Format: [lat, lng, intensity] where intensity is optional
          points.push([lat, lng, Math.random() * heatmapIntensity]);
        }
      } else if (type === 'clusters') {
        // Create several clusters
        const centers = [
          [51.505, -0.09], // Center
          [51.52, -0.08], // North
          [51.49, -0.11], // Southwest
          [51.51, -0.05], // Northeast
        ];

        // Generate points around each center
        centers.forEach((center) => {
          const clusterSize = count / centers.length;
          for (let i = 0; i < clusterSize; i++) {
            // Closer to the center = higher probability
            const distance = Math.random() * Math.random() * 0.025; // Squared
random for center bias
            const angle = Math.random() * 2 * Math.PI;
            const lat = center[0] + Math.cos(angle) * distance;
            const lng = center[1] + Math.sin(angle) * distance;

            points.push([lat, lng, Math.random() * heatmapIntensity]);
          }
        });
      } else if (type === 'path') {
        // Create a line/path pattern
        const pathPoints = [
          [51.505, -0.12],
          [51.51, -0.11],
          [51.52, -0.09],
          [51.51, -0.06],
          [51.49, -0.05],
        ];

        // Generate points along the path
        for (let i = 0; i < pathPoints.length - 1; i++) {
          const start = pathPoints[i];
          const end = pathPoints[i + 1];
          const pointsInSegment = count / (pathPoints.length - 1);

          for (let j = 0; j < pointsInSegment; j++) {
```

```
            const ratio = j / pointsInSegment;
            // Linear interpolation between points with random noise
            const lat =
              start[0] +
              (end[0] - start[0]) * ratio +
              (Math.random() - 0.5) * 0.01;
            const lng =
              start[1] +
              (end[1] - start[1]) * ratio +
              (Math.random() - 0.5) * 0.01;

            points.push([lat, lng, Math.random() * heatmapIntensity]);
          }
        }
      }

      return points;
    };

    setHeatmapPoints(generateHeatmapData(dataSet));
    setLoading(false);
  }, [dataSet, heatmapIntensity]);

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Heatmap Visualization</h3>
        {loading ? (
          <p>Generating heatmap data...</p>
        ) : (
          <p>Showing heatmap with {heatmapPoints.length} points.</p>
        )}

        <div
          style={{
            display: 'flex',
            flexWrap: 'wrap',
            gap: '20px',
            marginBottom: '15px',
          }}
        >
          <div>
            <div>Data Pattern:</div>
            <div>
              <select
                value={dataSet}
                onChange={(e) => setDataSet(e.target.value)}
                style={{ marginTop: '5px' }}
              >
                <option value="uniform">Uniform Distribution</option>
                <option value="clusters">Clustered Data</option>
                <option value="path">Path Pattern</option>
              </select>
            </div>
```

```
            </div>

            <div>
              <div>Radius: {heatmapRadius}px</div>
              <input
                type="range"
                min="10"
                max="50"
                value={heatmapRadius}
                onChange={(e) => setHeatmapRadius(Number(e.target.value))}
              />
            </div>

            <div>
              <div>Blur: {heatmapBlur}px</div>
              <input
                type="range"
                min="5"
                max="25"
                value={heatmapBlur}
                onChange={(e) => setHeatmapBlur(Number(e.target.value))}
              />
            </div>

            <div>
              <div>Intensity: {heatmapIntensity.toFixed(1)}</div>
              <input
                type="range"
                min="0.1"
                max="1.0"
                step="0.1"
                value={heatmapIntensity}
                onChange={(e) => setHeatmapIntensity(Number(e.target.value))}
              />
            </div>
          </div>
        </div>

        <MapContainer
          center={[51.505, -0.09]}
          zoom={13}
          style={{ height: '500px', width: '100%' }}
        >
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />

          {!loading && heatmapPoints.length > 0 && (
            <HeatmapLayer
              points={heatmapPoints}
              intensity={heatmapIntensity}
              radius={heatmapRadius}
```

```
              blur={heatmapBlur}
            />
          )}
        </MapContainer>

        <div style={{ marginTop: '15px' }}>
          <h4>Heatmap Use Cases</h4>
          <ul>
            <li>
              <strong>Population Density</strong>: Show where people live or
              congregate
            </li>
            <li>
              <strong>Activity Hotspots</strong>: Visualize user interactions,
              clicks, or events
            </li>
            <li>
              <strong>Environmental Data</strong>: Display temperature, pollution,
              or rainfall patterns
            </li>
            <li>
              <strong>Business Analytics</strong>: Identify high-traffic areas or
              customer concentrations
            </li>
          </ul>

          <h4>Implementation Notes</h4>
          <ul>
            <li>
              Use <code>leaflet.heat</code> plugin for easy heatmap integration
            </li>
            <li>
              Points can include intensity values as a third parameter:{' '}
              <code>[lat, lng, intensity]</code>
            </li>
            <li>Adjust radius and blur for different visual effects</li>
            <li>
              Consider using a subdued basemap to make the heatmap more visible
            </li>
            <li>
              For large datasets (50,000+ points), consider pre-processing or
              aggregating data on the server
            </li>
          </ul>
        </div>
      </div>
    );
};

export default HeatmapExample;
```

Custom Tile Layers

You can create and use custom tile layers for specialized map applications:

```
import React, { useState } from 'react';
import {
  MapContainer,
  TileLayer,
  LayersControl,
  ImageOverlay,
} from 'react-leaflet';
import L from 'leaflet';

const { BaseLayer } = LayersControl;

const CustomTileLayersExample = () => {
  const [opacity, setOpacity] = useState(1.0);
  const [selectedLayer, setSelectedLayer] = useState('osm');

  // Example of a custom tile URL template function
  const getStamenTonerUrl = (style) => {
    return `https://stamen-tiles-{s}.a.ssl.fastly.net/${style}/{z}/{x}/{y}
{r}.png`;
  };

  // Custom tile layers
  const tileLayers = {
    osm: {
      name: 'OpenStreetMap',
      url: 'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
      attribution:
        '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
      maxZoom: 19,
    },
    toner: {
      name: 'Stamen Toner',
      url: getStamenTonerUrl('toner'),
      attribution:
        'Map tiles by <a href="http://stamen.com">Stamen Design</a>, <a
href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a> &mdash; Map data
&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors',
      maxZoom: 18,
    },
    watercolor: {
      name: 'Stamen Watercolor',
      url: getStamenTonerUrl('watercolor'),
      attribution:
        'Map tiles by <a href="http://stamen.com">Stamen Design</a>, <a
href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a> &mdash; Map data
&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors',
      maxZoom: 16,
    },
```

```
    terrain: {
      name: 'Stamen Terrain',
      url: getStamenTonerUrl('terrain'),
      attribution:
        'Map tiles by <a href="http://stamen.com">Stamen Design</a>, <a
href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a> &mdash; Map data
&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors',
      maxZoom: 17,
    },
    satellite: {
      name: 'ESRI World Imagery',
      url:
'https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile
/{z}/{y}/{x}',
      attribution:
        'Tiles &copy; Esri &mdash; Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye,
Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community',
      maxZoom: 19,
    },
  };

  // Custom style for selected layer indicator
  const getLayerButtonStyle = (layer) => {
    return {
      padding: '6px 12px',
      margin: '0 5px 5px 0',
      backgroundColor: layer === selectedLayer ? '#3388ff' : '#f8f9fa',
      color: layer === selectedLayer ? 'white' : 'black',
      border: '1px solid #ddd',
      borderRadius: '4px',
      cursor: 'pointer',
    };
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Custom Tile Layers</h3>
        <p>Select different tile layers to change the map appearance:</p>

        <div
          style={{ display: 'flex', flexWrap: 'wrap', marginBottom: '10px' }}
        >
          {Object.keys(tileLayers).map((key) => (
            <button
              key={key}
              style={getLayerButtonStyle(key)}
              onClick={() => setSelectedLayer(key)}
            >
              {tileLayers[key].name}
            </button>
          ))}
        </div>
```

```
        <div style={{ marginBottom: '10px' }}>
          <label>
            Layer Opacity: {opacity.toFixed(1)}
            <input
              type="range"
              min="0.1"
              max="1"
              step="0.1"
              value={opacity}
              onChange={(e) => setOpacity(Number(e.target.value))}
              style={{ marginLeft: '10px', width: '200px' }}
            />
          </label>
        </div>
      </div>

      <MapContainer
        center={[51.505, -0.09]}
        zoom={13}
        style={{ height: '500px', width: '100%' }}
      >
        <LayersControl position="topright">
          {Object.keys(tileLayers).map((key) => (
            <BaseLayer
              key={key}
              name={tileLayers[key].name}
              checked={key === selectedLayer}
            >
              <TileLayer
                url={tileLayers[key].url}
                attribution={tileLayers[key].attribution}
                maxZoom={tileLayers[key].maxZoom}
                opacity={opacity}
              />
            </BaseLayer>
          ))}
        </LayersControl>

        {/* Example of custom image overlay */}
        <ImageOverlay
          url="https://maps.lib.utexas.edu/maps/historical/newark_nj_1922.jpg"
          bounds={[
            [40.712216, -74.22655],
            [40.773941, -74.12544],
          ]}
          opacity={0.5}
          zIndex={10}
        />
      </MapContainer>

      <div style={{ marginTop: '15px' }}>
        <h4>Creating Your Own Tile Layers</h4>
        <p>There are several ways to create custom tile layers:</p>
```

```
<ul>
  <li>
    <strong>Existing Providers</strong>: Use free or commercial map tile
    services (Mapbox, Stamen, etc.)
  </li>
  <li>
    <strong>Self-Hosted</strong>: Generate and host your own tiles with
    tools like <code>gdal2tiles</code> or <code>TileMill</code>
  </li>
  <li>
    <strong>Vector Tiles</strong>: Use vector tile services with custom
    styling (Mapbox GL, MapLibre)
  </li>
  <li>
    <strong>WMS Services</strong>: Connect to Web Map Services for
    specialized data layers
  </li>
</ul>

<h4>Custom Tile URL Format</h4>
<p>The standard URL template format includes these placeholders:</p>
<ul>
  <li>
    <code>{'{z}'}</code>: The zoom level
  </li>
  <li>
    <code>{'{x}'}</code> and <code>{'{y}'}</code>: The tile coordinates
  </li>
  <li>
    <code>{'{s}'}</code>: The subdomain (for load balancing across
    servers)
  </li>
  <li>
    <code>{'{r}'}</code>: Used for high-DPI ("retina") tiles, typically
    '@2x'
  </li>
</ul>

<h4>Performance Considerations</h4>
<ul>
  <li>
    Use appropriate max/min zoom levels to prevent unnecessary tile
    requests
  </li>
  <li>Consider adding tile caching for frequently accessed areas</li>
  <li>
    Adjust the opacity for overlay tile layers to maintain readability
  </li>
  <li>
    If using multiple overlay layers, manage their z-index and opacity
    carefully
  </li>
</ul>
```

```
      </div>
    </div>
  );
};

export default CustomTileLayersExample;
```

Integration with External Data Sources and APIs

Connecting your Leaflet map to external data sources is common for real-world applications:

```
import React, { useState, useEffect } from 'react';
import {
  MapContainer,
  TileLayer,
  Marker,
  Popup,
  CircleMarker,
  useMap,
} from 'react-leaflet';
import L from 'leaflet';

// Component to trigger API requests when the map view changes
const MapViewMonitor = ({ onViewChange }) => {
  const map = useMap();

  useEffect(() => {
    // Function to handle map moveend event
    const handleMoveEnd = () => {
      const center = map.getCenter();
      const zoom = map.getZoom();
      const bounds = map.getBounds();

      onViewChange({
        center: [center.lat, center.lng],
        zoom,
        bounds: {
          north: bounds.getNorth(),
          south: bounds.getSouth(),
          east: bounds.getEast(),
          west: bounds.getWest(),
        },
      });
    };

    // Register event listener
    map.on('moveend', handleMoveEnd);

    // Trigger initial load
    handleMoveEnd();
```

```
      // Clean up
      return () => {
        map.off('moveend', handleMoveEnd);
      };
    }, [map, onViewChange]);

    return null;
  };

  const ExternalDataExample = () => {
    const [loading, setLoading] = useState(false);
    const [error, setError] = useState(null);
    const [dataSource, setDataSource] = useState('openweather');
    const [dataPoints, setDataPoints] = useState([]);
    const [mapView, setMapView] = useState(null);

    // API keys - in a real app, these would be environment variables
    // You should obtain your own API keys for these services
    const API_KEYS = {
      openweather: 'YOUR_OPENWEATHER_API_KEY',
      geoapify: 'YOUR_GEOAPIFY_API_KEY',
      foursquare: 'YOUR_FOURSQUARE_API_KEY',
    };

    // Mock data for demonstration (when API keys aren't provided)
    const MOCK_DATA = {
      openweather: [
        {
          id: 1,
          name: 'London',
          position: [51.505, -0.09],
          temp: 12.5,
          humidity: 76,
          windSpeed: 4.2,
        },
        {
          id: 2,
          name: 'Cambridge',
          position: [52.205, 0.119],
          temp: 11.8,
          humidity: 81,
          windSpeed: 3.7,
        },
        {
          id: 3,
          name: 'Oxford',
          position: [51.752, -1.257],
          temp: 12.1,
          humidity: 78,
          windSpeed: 3.9,
        },
      ],
      geoapify: [
        {
```

```
        id: 1,
        name: 'British Museum',
        position: [51.519, -0.127],
        category: 'museum',
        address: 'Great Russell St, London',
      },
      {
        id: 2,
        name: 'Hyde Park',
        position: [51.507, -0.166],
        category: 'park',
        address: 'London',
      },
      {
        id: 3,
        name: 'London Bridge',
        position: [51.508, -0.087],
        category: 'landmark',
        address: 'London Bridge, London',
      },
    ],
    foursquare: [
      {
        id: 1,
        name: 'The Ivy',
        position: [51.511, -0.128],
        category: 'restaurant',
        rating: 4.6,
      },
      {
        id: 2,
        name: 'The Ritz',
        position: [51.507, -0.142],
        category: 'hotel',
        rating: 4.8,
      },
      {
        id: 3,
        name: 'Harrods',
        position: [51.499, -0.163],
        category: 'shop',
        rating: 4.5,
      },
    ],
  };

  // Load data when map view changes or data source changes
  useEffect(() => {
    if (!mapView) return;

    // Determine function to fetch data based on selected source
    const fetchData = async () => {
      setLoading(true);
      setError(null);
```

```
      try {
        let data = [];

        // In a real application, you would use actual API calls instead of these
mock implementations
        switch (dataSource) {
          case 'openweather':
            // OpenWeather API for weather data (mocked)
            // Actual implementation would use:
            // const response = await
fetch(`https://api.openweathermap.org/data/2.5/find?
lat=${mapView.center[0]}&lon=${mapView.center[1]}&cnt=10&units=metric&appid=${API_
KEYS.openweather}`);
            // data = await response.json();

            // Using mock data for demonstration
            data = MOCK_DATA.openweather;
            break;

          case 'geoapify':
            // Geoapify for POI data (mocked)
            // Actual implementation would use:
            // const response = await fetch(`https://api.geoapify.com/v2/places?
categories=commercial.shopping&filter=rect:${mapView.bounds.west},${mapView.bounds
.south},${mapView.bounds.east},${mapView.bounds.north}&limit=20&apiKey=${API_KEYS.
geoapify}`);
            // data = await response.json();

            // Using mock data for demonstration
            data = MOCK_DATA.geoapify;
            break;

          case 'foursquare':
            // Foursquare for venue data (mocked)
            // Actual implementation would use:
            // const response = await
fetch(`https://api.foursquare.com/v3/places/search?
ll=${mapView.center[0]},${mapView.center[1]}&radius=10000&limit=20`, {
            //   headers: {
            //     'Authorization': API_KEYS.foursquare
            //   }
            // });
            // data = await response.json();

            // Using mock data for demonstration
            data = MOCK_DATA.foursquare;
            break;

          default:
            data = [];
        }

        setDataPoints(data);
```

```
        } catch (err) {
          console.error('Error fetching data:', err);
          setError(`Failed to load data: ${err.message}`);
        } finally {
          setLoading(false);
        }
      };

      fetchData();
    }, [mapView, dataSource]);

    // Get marker icon based on data source
    const getMarkerIcon = (item) => {
      let iconHtml = '';
      let size = [30, 30];

      switch (dataSource) {
        case 'openweather':
          // Weather icon
          iconHtml = `
            <div style="
              background-color: rgba(66, 165, 245, 0.8);
              color: white;
              border-radius: 50%;
              text-align: center;
              font-weight: bold;
              width: 30px;
              height: 30px;
              line-height: 30px;
            ">
              ${Math.round(item.temp)}°
            </div>
          `;
          break;

        case 'geoapify':
          // POI icon
          let color = '#9c27b0'; // Default
          if (item.category === 'museum') color = '#f44336';
          if (item.category === 'park') color = '#4caf50';
          if (item.category === 'landmark') color = '#ff9800';

          iconHtml = `
            <div style="
              background-color: ${color};
              color: white;
              border-radius: 50%;
              text-align: center;
              font-weight: bold;
              width: 30px;
              height: 30px;
              line-height: 30px;
              font-size: 12px;
            ">
```

```
            ${item.category.substring(0, 1).toUpperCase()}
          </div>
        `;
        break;

      case 'foursquare':
        // Venue icon with rating
        const ratingColor =
          item.rating >= 4.5
            ? '#4caf50'
            : item.rating >= 4.0
            ? '#ff9800'
            : '#f44336';

        iconHtml = `
          <div style="
            background-color: white;
            color: black;
            border: 2px solid ${ratingColor};
            border-radius: 4px;
            text-align: center;
            font-weight: bold;
            width: 36px;
            height: 20px;
            line-height: 20px;
            font-size: 12px;
          ">
            ${item.rating}
          </div>
        `;
        size = [36, 20];
        break;

      default:
        // Default marker
        return null;
    }

    return L.divIcon({
      html: iconHtml,
      className: 'custom-div-icon',
      iconSize: size,
      iconAnchor: [size[0] / 2, size[1] / 2],
    });
  };

  // Render popup content based on data source
  const renderPopupContent = (item) => {
    switch (dataSource) {
      case 'openweather':
        return (
          <div>
            <h3>{item.name}</h3>
            <p>
```

```
              <strong>Temperature:</strong> {item.temp}°C
              <br />
              <strong>Humidity:</strong> {item.humidity}%<br />
              <strong>Wind Speed:</strong> {item.windSpeed} m/s
            </p>
          </div>
        );

      case 'geoapify':
        return (
          <div>
            <h3>{item.name}</h3>
            <p>
              <strong>Category:</strong> {item.category}
              <br />
              <strong>Address:</strong> {item.address}
            </p>
          </div>
        );

      case 'foursquare':
        return (
          <div>
            <h3>{item.name}</h3>
            <p>
              <strong>Category:</strong> {item.category}
              <br />
              <strong>Rating:</strong> {item.rating}
            </p>
          </div>
        );

      default:
        return <div>{JSON.stringify(item)}</div>;
    }
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>External Data Integration</h3>
        <p>Select a data source to display on the map:</p>

        <div style={{ marginBottom: '10px' }}>
          <select
            value={dataSource}
            onChange={(e) => setDataSource(e.target.value)}
            style={{ marginRight: '10px' }}
          >
            <option value="openweather">Weather Data (OpenWeather)</option>
            <option value="geoapify">Points of Interest (Geoapify)</option>
            <option value="foursquare">Venues (Foursquare)</option>
          </select>
```

```
            {loading && <span>Loading data...</span>}
            {error && <span style={{ color: 'red' }}>{error}</span>}
          </div>

          <p>
            <strong>Note:</strong> This example uses mock data for demonstration.
            In a real application, you would need to obtain API keys for the
            respective services.
          </p>
        </div>

        <MapContainer
          center={[51.505, -0.09]}
          zoom={12}
          style={{ height: '500px', width: '100%' }}
        >
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />

          <MapViewMonitor onViewChange={setMapView} />

          {/* Render data points based on selected source */}
          {dataPoints.map((item) => {
            const icon = getMarkerIcon(item);

            return icon ? (
              <Marker key={item.id} position={item.position} icon={icon}>
                <Popup>{renderPopupContent(item)}</Popup>
              </Marker>
            ) : (
              <CircleMarker
                key={item.id}
                center={item.position}
                radius={8}
                pathOptions={{
                  fillColor: '#3388ff',
                  color: '#3388ff',
                  fillOpacity: 0.7,
                  weight: 2,
                }}
              >
                <Popup>{renderPopupContent(item)}</Popup>
              </CircleMarker>
            );
          })}
        </MapContainer>

        <div style={{ marginTop: '15px' }}>
          <h4>Integration Strategies</h4>
          <ul>
            <li>
```

```
            <strong>Live API Calls</strong>: Fetch data as the user interacts
            with the map (e.g., on pan/zoom)
          </li>
          <li>
            <strong>WebSockets</strong>: For real-time data that changes
            frequently (e.g., vehicle tracking)
          </li>
          <li>
            <strong>Pre-loaded Data</strong>: Load data once and filter/display
            it based on the map view
          </li>
          <li>
            <strong>Incremental Loading</strong>: Load data in chunks as needed
            to maintain performance
          </li>
        </ul>

        <h4>Best Practices</h4>
        <ul>
          <li>
            Implement <strong>caching</strong> to reduce redundant API calls
          </li>
          <li>
            Add <strong>throttling</strong> to prevent too many requests during
            continuous map movements
          </li>
          <li>
            Include <strong>loading indicators</strong> to provide feedback
            during data fetching
          </li>
          <li>
            Handle <strong>error states</strong> gracefully with appropriate
            user feedback
          </li>
          <li>
            Keep API keys <strong>secure</strong> by using environment variables
            and server-side proxies
          </li>
          <li>
            Consider <strong>data transformations</strong> to convert API
            responses into map-friendly formats
          </li>
        </ul>
      </div>
    </div>
  );
};

export default ExternalDataExample;
```

Performance Optimization for Large Datasets

Handling large datasets requires special techniques to maintain performance:

```
import React, { useState, useEffect, useCallback, useMemo } from 'react';
import { MapContainer, TileLayer, CircleMarker, useMap } from 'react-leaflet';
import MarkerClusterGroup from 'react-leaflet-cluster';

// Component to monitor visible part of the map
const VisibleAreaMonitor = ({ onVisibleAreaChange, minZoom }) => {
  const map = useMap();

  const updateVisibleArea = useCallback(() => {
    const bounds = map.getBounds();
    const zoom = map.getZoom();

    onVisibleAreaChange({
      bounds,
      zoom,
    });
  }, [map, onVisibleAreaChange]);

  useEffect(() => {
    // Update on map movement and zoom
    map.on('moveend', updateVisibleArea);
    map.on('zoomend', updateVisibleArea);

    // Initial update
    updateVisibleArea();

    return () => {
      map.off('moveend', updateVisibleArea);
      map.off('zoomend', updateVisibleArea);
    };
  }, [map, updateVisibleArea]);

  // Enforce minimum zoom level
  useEffect(() => {
    if (map.getZoom() < minZoom) {
      map.setZoom(minZoom);
    }
  }, [map, minZoom]);

  return null;
};

const LargeDatasetExample = () => {
  const [allPoints, setAllPoints] = useState([]);
  const [visiblePoints, setVisiblePoints] = useState([]);
  const [visibleArea, setVisibleArea] = useState(null);
  const [clusteringEnabled, setClusteringEnabled] = useState(true);
  const [simplificationLevel, setSimplificationLevel] = useState('medium');
  const [loading, setLoading] = useState(true);
  const [showStats, setShowStats] = useState(true);
  const [renderTime, setRenderTime] = useState(0);
```

```
  // Configuration
  const MIN_ZOOM_LEVEL = 10; // Restrict zoom to avoid rendering too many points
  const MAX_VISIBLE_POINTS = 2000; // Limit for performance

  // Generate a large dataset on mount
  useEffect(() => {
    const generateLargeDataset = () => {
      setLoading(true);

      // Create worker for better performance
      const worker = new Worker(
        URL.createObjectURL(
          new Blob(
            [
              `
              self.onmessage = function(e) {
                const count = e.data.count;
                const center = e.data.center;
                const radius = e.data.radius;

                const points = [];

                for (let i = 0; i < count; i++) {
                  // Create points in a circle around center with Gaussian
distribution
                  const angle = Math.random() * Math.PI * 2;
                  const r = Math.sqrt(-2 * Math.log(Math.random())) * radius / 2;

                  points.push({
                    id: i,
                    position: [
                      center[0] + r * Math.cos(angle),
                      center[1] + r * Math.sin(angle)
                    ],
                    value: Math.random() * 100,
                    category: Math.floor(Math.random() * 4)
                  });
                }

                self.postMessage(points);
              }
            `,
            ],
            { type: 'application/javascript' }
          )
        )
      );

      worker.onmessage = (e) => {
        setAllPoints(e.data);
        setLoading(false);
      };
```

```
      // Generate 50,000 points around London
      worker.postMessage({
        count: 50000,
        center: [51.505, -0.09],
        radius: 0.5, // ~50km
      });

      return () => worker.terminate();
    };

    generateLargeDataset();
  }, []);

  // Filter points based on visible area and simplification level
  useEffect(() => {
    if (!visibleArea || !allPoints.length) return;

    const startTime = performance.now();

    // Function to check if a point is in bounds
    const isInBounds = (point) => {
      const { bounds } = visibleArea;
      const [lat, lng] = point.position;
      return (
        lat >= bounds.getSouth() &&
        lat <= bounds.getNorth() &&
        lng >= bounds.getWest() &&
        lng <= bounds.getEast()
      );
    };

    // Get points in current bounds
    let pointsInBounds = allPoints.filter(isInBounds);

    // Apply simplification based on level
    let simplifiedPoints;

    if (simplificationLevel === 'high') {
      // Show all points in bounds
      simplifiedPoints = pointsInBounds;
    } else {
      // Get a subset of points
      const samplingRate = simplificationLevel === 'low' ? 10 : 4; // 1/10 or 1/4
of points

      if (pointsInBounds.length > MAX_VISIBLE_POINTS) {
        const interval = Math.max(
          1,
          Math.floor(pointsInBounds.length / MAX_VISIBLE_POINTS) * samplingRate
        );
        simplifiedPoints = pointsInBounds.filter(
          (_, index) => index % interval === 0
        );
      } else {
```

```
        simplifiedPoints = pointsInBounds.filter(
          (_, index) => index % samplingRate === 0
        );
      }
    }

    setVisiblePoints(simplifiedPoints);
    setRenderTime(performance.now() - startTime);
  }, [visibleArea, allPoints, simplificationLevel]);

  // Get color based on point category
  const getCategoryColor = useCallback((category) => {
    const colors = ['#e53935', '#43a047', '#1e88e5', '#fdd835'];
    return colors[category % colors.length];
  }, []);

  // Memoize marker style to avoid recalculations
  const getMarkerStyle = useCallback(
    (point) => {
      return {
        radius: 4,
        fillColor: getCategoryColor(point.category),
        color: getCategoryColor(point.category),
        weight: 1,
        opacity: 0.8,
        fillOpacity: 0.6,
      };
    },
    [getCategoryColor]
  );

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Large Dataset Optimization</h3>

        {loading ? (
          <p>Generating large dataset (50,000 points)...</p>
        ) : (
          <div>
            <p>
              Total Points: {allPoints.length.toLocaleString()}
              <br />
              Visible Points: {visiblePoints.length.toLocaleString()}
              <br />
              {showStats && <span>Render Time: {renderTime.toFixed(2)}ms</span>}
            </p>

            <div
              style={{
                display: 'flex',
                flexWrap: 'wrap',
                gap: '15px',
                marginBottom: '10px',
```

```
            }}
          >
            <div>
              <label>
                <input
                  type="checkbox"
                  checked={clusteringEnabled}
                  onChange={(e) => setClusteringEnabled(e.target.checked)}
                />
                Enable clustering
              </label>
            </div>

            <div>
              <label>Detail Level: </label>
              <select
                value={simplificationLevel}
                onChange={(e) => setSimplificationLevel(e.target.value)}
              >
                <option value="low">Low (Best Performance)</option>
                <option value="medium">Medium (Balanced)</option>
                <option value="high">High (Most Detailed)</option>
              </select>
            </div>

            <div>
              <label>
                <input
                  type="checkbox"
                  checked={showStats}
                  onChange={(e) => setShowStats(e.target.checked)}
                />
                Show performance stats
              </label>
            </div>
          </div>

          <p>
            <strong>Note:</strong> Zoom is restricted to level{' '}
            {MIN_ZOOM_LEVEL}+ for performance.
          </p>
        </div>
      )}
    </div>

    <MapContainer
      center={[51.505, -0.09]}
      zoom={11}
      minZoom={MIN_ZOOM_LEVEL}
      style={{ height: '500px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
```

```
  href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        <VisibleAreaMonitor
          onVisibleAreaChange={setVisibleArea}
          minZoom={MIN_ZOOM_LEVEL}
        />

        {/* Render points */}
        {!loading &&
          visiblePoints.length > 0 &&
          (clusteringEnabled ? (
            <MarkerClusterGroup
              chunkedLoading
              maxClusterRadius={80}
              spiderfyOnMaxZoom={false}
            >
              {visiblePoints.map((point) => (
                <CircleMarker
                  key={point.id}
                  center={point.position}
                  pathOptions={getMarkerStyle(point)}
                />
              ))}
            </MarkerClusterGroup>
          ) : (
            <>
              {visiblePoints.map((point) => (
                <CircleMarker
                  key={point.id}
                  center={point.position}
                  pathOptions={getMarkerStyle(point)}
                />
              ))}
            </>
          ))}
      </MapContainer>

      <div style={{ marginTop: '15px' }}>
        <h4>Performance Optimization Techniques</h4>
        <ul>
          <li>
            <strong>Clustering</strong>: Group nearby points together to reduce
            DOM elements
          </li>
          <li>
            <strong>Point Sampling</strong>: Only render a subset of points
            based on zoom level
          </li>
          <li>
            <strong>Bounds Filtering</strong>: Only render points that are
            visible in the current view
          </li>
          <li>
```

```
            <strong>Zoom Restrictions</strong>: Prevent zooming out too far to
            avoid rendering too many points
          </li>
          <li>
            <strong>WebWorkers</strong>: Use background threads for data
            processing and filtering
          </li>
          <li>
            <strong>Canvas Rendering</strong>: Use canvas instead of SVG for
            better performance with many points
          </li>
          <li>
            <strong>Virtualization</strong>: Only create DOM elements for
            visible points
          </li>
        </ul>

        <h4>When to Use Which Technique</h4>
        <ul>
          <li>
            For 100-1,000 points: Standard rendering with clustering is usually
            sufficient
          </li>
          <li>
            For 1,000-10,000 points: Add bounds filtering and consider canvas
            rendering
          </li>
          <li>
            For 10,000-100,000 points: Implement point sampling, WebWorkers, and
            all above techniques
          </li>
          <li>
            For >100,000 points: Server-side clustering or aggregation is
            recommended
          </li>
        </ul>
      </div>
    </div>
  );
};

export default LargeDatasetExample;
```

Handling Specific Challenges

Working with Real-time Data on Maps

```
import React, { useState, useEffect, useRef } from 'react';
import {
  MapContainer,
  TileLayer,
```

```
  Marker,
  Polyline,
  Tooltip,
  useMap,
} from 'react-leaflet';
import L from 'leaflet';

// Component to keep markers in view
const AutoPan = ({ positions, follow }) => {
  const map = useMap();

  useEffect(() => {
    if (!follow || positions.length === 0) return;

    // Get the last position (most recent)
    const lastPosition = positions[positions.length - 1].position;

    // Pan the map to the most recent position
    map.panTo(lastPosition);
  }, [map, positions, follow]);

  return null;
};

const RealTimeDataExample = () => {
  const [vehicles, setVehicles] = useState([]);
  const [selectedVehicle, setSelectedVehicle] = useState(null);
  const [followSelected, setFollowSelected] = useState(true);
  const [simulationSpeed, setSimulationSpeed] = useState(1);
  const [simulationRunning, setSimulationRunning] = useState(true);
  const simulationInterval = useRef(null);

  // Create our vehicle simulation
  useEffect(() => {
    // Initial vehicle data
    const initialVehicles = [
      {
        id: 'v1',
        name: 'Bus 42',
        type: 'bus',
        color: '#e53935',
        position: [51.505, -0.09],
        speed: 30,
        heading: 45,
        history: [],
        route: [
          [51.505, -0.09],
          [51.51, -0.09],
          [51.51, -0.08],
          [51.515, -0.06],
          [51.52, -0.05],
          [51.525, -0.06],
        ],
      },
```

```
      {
        id: 'v2',
        name: 'Taxi 007',
        type: 'taxi',
        color: '#ffeb3b',
        position: [51.5, -0.1],
        speed: 45,
        heading: 90,
        history: [],
        route: [
          [51.5, -0.1],
          [51.5, -0.08],
          [51.495, -0.06],
          [51.49, -0.04],
          [51.485, -0.02],
        ],
      },
      {
        id: 'v3',
        name: 'Delivery',
        type: 'truck',
        color: '#43a047',
        position: [51.49, -0.11],
        speed: 25,
        heading: 0,
        history: [],
        route: [
          [51.49, -0.11],
          [51.485, -0.11],
          [51.48, -0.105],
          [51.475, -0.1],
          [51.47, -0.095],
        ],
      },
    ];

    setVehicles(initialVehicles);

    // Set initial selected vehicle
    setSelectedVehicle(initialVehicles[0].id);

    // Clean up
    return () => {
      if (simulationInterval.current) {
        clearInterval(simulationInterval.current);
      }
    };
  }, []);

  // Run the simulation
  useEffect(() => {
    if (!vehicles.length) return;

    if (simulationRunning) {
```

```
        if (simulationInterval.current) {
          clearInterval(simulationInterval.current);
        }

        simulationInterval.current = setInterval(() => {
          setVehicles((prevVehicles) => {
            return prevVehicles.map((vehicle) => {
              // Store current position in history
              const history = [
                ...vehicle.history,
                {
                  position: [...vehicle.position],
                  timestamp: new Date().toISOString(),
                },
              ];

              // Only keep the last 100 positions
              if (history.length > 100) {
                history.shift();
              }

              // Calculate next position along route
              const route = vehicle.route;
              const currentRouteIndex = route.findIndex(
                (point) =>
                  Math.abs(point[0] - vehicle.position[0]) < 0.001 &&
                  Math.abs(point[1] - vehicle.position[1]) < 0.001
              );

              // If at the end of route, reverse direction
              let nextRouteIndex;
              if (currentRouteIndex === route.length - 1) {
                nextRouteIndex = currentRouteIndex - 1;
              } else if (currentRouteIndex === 0) {
                nextRouteIndex = 1;
              } else if (currentRouteIndex === -1) {
                // Not exactly on a route point, find closest
                nextRouteIndex = 1;
              } else {
                nextRouteIndex = currentRouteIndex + 1;
              }

              // Get next target
              const nextTarget = route[nextRouteIndex];

              // Calculate direction and distance to next target
              const dx = nextTarget[1] - vehicle.position[1];
              const dy = nextTarget[0] - vehicle.position[0];
              const distance = Math.sqrt(dx * dx + dy * dy);

              // Calculate heading (in degrees)
              const heading = Math.atan2(dx, dy) * (180 / Math.PI);

              // Calculate step size based on speed and simulation speed
```

```
            // Convert km/h to degree/s (very rough approximation)
            const stepSize = (vehicle.speed / 3600) * 0.001 * simulationSpeed;

            // Calculate new position
            let newPosition;
            if (distance <= stepSize) {
              // Reached (or very close to) the target point
              newPosition = [...nextTarget];
            } else {
              // Move towards target
              const ratio = stepSize / distance;
              newPosition = [
                vehicle.position[0] + dy * ratio,
                vehicle.position[1] + dx * ratio,
              ];
            }

            return {
              ...vehicle,
              position: newPosition,
              heading,
              history,
            };
          });
        });
      }, 1000 / simulationSpeed); // Adjust interval based on speed
    } else if (simulationInterval.current) {
      clearInterval(simulationInterval.current);
    }

    return () => {
      if (simulationInterval.current) {
        clearInterval(simulationInterval.current);
      }
    };
  }, [vehicles, simulationRunning, simulationSpeed]);

  // Create marker icon based on vehicle type and heading
  const createVehicleIcon = (vehicle) => {
    // Create a path for the vehicle icon based on type
    let iconPath;

    switch (vehicle.type) {
      case 'bus':
        iconPath =
          'M4,16c0,1.1,0.9,2,2,2h12c1.1,0,2-0.9,2-2V6c0-1.1-0.9-2-2-
2H6C4.9,4,4,4.9,4,6V16z M18,16H6V6h12V16z M7,14h2v2H7V14z M15,14h2v2h-2V14z
M7,11h10v2H7V11z M7,8h10v2H7V8z';
        break;
      case 'taxi':
        iconPath =

'M18.92,6.01C18.72,5.42,18.16,5,17.5,5H15V3H9v2H6.5C5.84,5,5.29,5.42,5.08,6.01L3,1
2v8c0,0.55,0.45,1,1,1h1c0.55,0,1-0.45,1-1v-1h12v1c0,0.55,0.45,1,1,1h1c0.55,0,1-
```

```
0.45,1-1v-8L18.92,6.01z M6.85,7h10.29l1.04,3H5.81L6.85,7z M19,17H5v-5h14V17z
M7.5,14.5c0.83,0,1.5,0.67,1.5,1.5S8.33,17.5,7.5,17.5S6,16.83,6,16S6.67,14.5,7.5,14
.5z M16.5,14.5c0.83,0,1.5,0.67,1.5,1.5s-0.67,1.5-1.5,1.5s-1.5-0.67-1.5-
1.5S15.67,14.5,16.5,14.5z';
        break;
      case 'truck':
        iconPath =
          'M18,18.5c0.83,0,1.5-0.67,1.5-1.5h-3C16.5,17.83,17.17,18.5,18,18.5z
M17,4h-4v8h6V7L17,4z M6,18.5c0.83,0,1.5-0.67,1.5-1.5h-
3C4.5,17.83,5.17,18.5,6,18.5z M20,6h-3l1,3h-7.79l-7.04-
3.74C3.54,5.01,3.76,5.61,4.4,6.05l5.6,3.95V16c0,0.55,0.45,1,1,1h9c0.55,0,1-0.45,1-
1V6z';
        break;
      default:
        iconPath =
          'M12,2C8.13,2,5,5.13,5,9c0,5.25,7,13,7,13s7-7.75,7-
13C19,5.13,15.87,2,12,2z M12,11.5c-1.38,0-2.5-1.12-2.5-2.5s1.12-2.5,2.5-
2.5s2.5,1.12,2.5,2.5S13.38,11.5,12,11.5z';
    }

    // Create an SVG with the path rotated according to heading
    const svgString = `
      <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0
24 24">
        <g transform="rotate(${vehicle.heading}, 12, 12)">
          <path fill="${vehicle.color}" d="${iconPath}"/>
        </g>
      </svg>
    `;

    // Convert SVG to data URL
    const svgUrl = `${btoa(svgString)}`;

    return L.icon({
      iconUrl: svgUrl,
      iconSize: [24, 24],
      iconAnchor: [12, 12],
      popupAnchor: [0, -12],
    });
  };

  // Get the selected vehicle's data
  const selectedVehicleData = vehicles.find((v) => v.id === selectedVehicle);

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Real-time Data Visualization</h3>
        <p>
          This example simulates real-time vehicle tracking with position
          history.
        </p>

        <div
```

```
        style={{
          display: 'flex',
          flexWrap: 'wrap',
          gap: '20px',
          marginBottom: '10px',
        }}
      >
        <div>
          <label>Select Vehicle: </label>
          <select
            value={selectedVehicle || ''}
            onChange={(e) => setSelectedVehicle(e.target.value)}
            style={{ marginLeft: '5px' }}
          >
            {vehicles.map((vehicle) => (
              <option key={vehicle.id} value={vehicle.id}>
                {vehicle.name}
              </option>
            ))}
          </select>
        </div>

        <div>
          <label>
            <input
              type="checkbox"
              checked={followSelected}
              onChange={(e) => setFollowSelected(e.target.checked)}
            />
            Follow selected vehicle
          </label>
        </div>

        <div>
          <label>Simulation Speed: </label>
          <select
            value={simulationSpeed}
            onChange={(e) => setSimulationSpeed(Number(e.target.value))}
            style={{ marginLeft: '5px' }}
          >
            <option value={0.5}>0.5x (Slow)</option>
            <option value={1}>1x (Normal)</option>
            <option value={2}>2x (Fast)</option>
            <option value={5}>5x (Very Fast)</option>
          </select>
        </div>

        <button
          onClick={() => setSimulationRunning(!simulationRunning)}
          style={{ padding: '2px 10px' }}
        >
          {simulationRunning ? 'Pause' : 'Resume'}
        </button>
      </div>
```

```
        {selectedVehicleData && (
          <div
            style={{
              backgroundColor: '#f5f5f5',
              padding: '10px',
              borderRadius: '4px',
              marginBottom: '10px',
            }}
          >
            <h4 style={{ margin: '0 0 8px 0' }}>{selectedVehicleData.name}</h4>
            <div>
              <strong>Current Position:</strong>{' '}
              {selectedVehicleData.position[0].toFixed(5)},{' '}
              {selectedVehicleData.position[1].toFixed(5)}
              <br />
              <strong>Speed:</strong> {selectedVehicleData.speed} km/h
              <br />
              <strong>Heading:</strong>{' '}
              {Math.round(selectedVehicleData.heading)}°<br />
              <strong>History Points:</strong>
{selectedVehicleData.history.length}
            </div>
          </div>
        )}
      </div>

      <MapContainer
        center={[51.505, -0.09]}
        zoom={13}
        style={{ height: '500px', width: '100%' }}
      >
        <TileLayer
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        />

        {/* Render all vehicles */}
        {vehicles.map((vehicle) => (
          <React.Fragment key={vehicle.id}>
            {/* Vehicle marker */}
            <Marker
              position={vehicle.position}
              icon={createVehicleIcon(vehicle)}
              eventHandlers={{
                click: () => setSelectedVehicle(vehicle.id),
              }}
            >
              <Tooltip direction="top" offset={[0, -10]} opacity={0.9}>
                <div>
                  <strong>{vehicle.name}</strong>
                  <br />
                  {vehicle.speed} km/h
```

```
                </div>
              </Tooltip>
            </Marker>

            {/* History trail (only for selected vehicle) */}
            {vehicle.id === selectedVehicle && vehicle.history.length > 1 && (
              <Polyline
                positions={vehicle.history.map((h) => h.position)}
                pathOptions={{
                  color: vehicle.color,
                  weight: 3,
                  opacity: 0.7,
                  dashArray: '5, 5',
                }}
              />
            )}

            {/* Route (only for selected vehicle) */}
            {vehicle.id === selectedVehicle && (
              <Polyline
                positions={vehicle.route}
                pathOptions={{
                  color: '#777',
                  weight: 2,
                  opacity: 0.5,
                }}
              />
            )}
          </React.Fragment>
        ))}

        {/* Auto-pan to follow selected vehicle */}
        {selectedVehicleData && (
          <AutoPan
            positions={selectedVehicleData.history}
            follow={followSelected}
          />
        )}
      </MapContainer>

      <div style={{ marginTop: '15px' }}>
        <h4>Real-time Data Strategies</h4>
        <ul>
          <li>
            <strong>Polling</strong>: Regular HTTP requests at set intervals
            (simplest approach)
          </li>
          <li>
            <strong>WebSockets</strong>: Bi-directional communication for
            immediate updates
          </li>
          <li>
            <strong>Server-Sent Events (SSE)</strong>: One-way server-to-client
            updates
```

```
            </li>
            <li>
              <strong>GraphQL Subscriptions</strong>: Real-time data with GraphQL
            </li>
          </ul>

          <h4>Implementation Considerations</h4>
          <ul>
            <li>
              Consider <strong>data volume</strong> - only send what's needed
              (position, heading, status)
            </li>
            <li>
              Implement <strong>interpolation</strong> for smoother movement
              between position updates
            </li>
            <li>
              Include <strong>timestamps</strong> with each update for accurate
              history
            </li>
            <li>
              Implement <strong>buffering</strong> to handle connection
              interruptions
            </li>
            <li>
              Use <strong>throttling</strong> to prevent excessive updates
            </li>
            <li>
              Consider <strong>battery impact</strong> for mobile devices
            </li>
          </ul>
        </div>
      </div>
    );
};

export default RealTimeDataExample;
```

Mobile Responsiveness and Touch Interactions

```
import React, { useState, useEffect } from 'react';
import {
  MapContainer,
  TileLayer,
  Marker,
  Popup,
  ZoomControl,
  useMap,
} from 'react-leaflet';
import L from 'leaflet';
```

```
// Component to detect touch support and map size
const DeviceDetector = ({ onDeviceInfoUpdate }) => {
  const map = useMap();

  useEffect(() => {
    // Detect touch capability
    const isTouchDevice =
      'ontouchstart' in window ||
      navigator.maxTouchPoints > 0 ||
      navigator.msMaxTouchPoints > 0;

    // Get map size
    const mapSize = map.getSize();

    // Detect device type based on screen width
    let deviceType = 'desktop';
    if (window.innerWidth <= 768) {
      deviceType = 'mobile';
    } else if (window.innerWidth <= 1024) {
      deviceType = 'tablet';
    }

    onDeviceInfoUpdate({
      isTouchDevice,
      mapSize,
      deviceType,
      screenWidth: window.innerWidth,
      screenHeight: window.innerHeight,
    });

    // Update on resize
    const handleResize = () => {
      const newMapSize = map.getSize();
      let newDeviceType = 'desktop';
      if (window.innerWidth <= 768) {
        newDeviceType = 'mobile';
      } else if (window.innerWidth <= 1024) {
        newDeviceType = 'tablet';
      }

      onDeviceInfoUpdate({
        isTouchDevice,
        mapSize: newMapSize,
        deviceType: newDeviceType,
        screenWidth: window.innerWidth,
        screenHeight: window.innerHeight,
      });
    };

    window.addEventListener('resize', handleResize);

    return () => {
      window.removeEventListener('resize', handleResize);
    };
```

```
  }, [map, onDeviceInfoUpdate]);

  return null;
};

const MobileResponsiveExample = () => {
  const [deviceInfo, setDeviceInfo] = useState(null);
  const [markerSize, setMarkerSize] = useState('medium'); // small, medium, large
  const [controlPosition, setControlPosition] = useState('topright');
  const [touchSimulation, setTouchSimulation] = useState(false);
  const [touchPoints, setTouchPoints] = useState([]);

  // Points of interest
  const pois = [
    {
      id: 1,
      name: 'London Eye',
      position: [51.503, -0.119],
      description: 'Giant Ferris wheel on the South Bank of the River Thames',
    },
    {
      id: 2,
      name: 'Tower of London',
      position: [51.508, -0.076],
      description: 'Historic castle on the north bank of the River Thames',
    },
    {
      id: 3,
      name: 'Hyde Park',
      position: [51.507, -0.165],
      description: 'Major park in central London',
    },
    {
      id: 4,
      name: 'British Museum',
      position: [51.519, -0.127],
      description: 'Public museum dedicated to human history, art and culture',
    },
    {
      id: 5,
      name: 'Buckingham Palace',
      position: [51.501, -0.142],
      description:
        'London residence and administrative headquarters of the monarch of the
United Kingdom',
    },
  ];

  // Create marker icon based on selected size
  const createMarkerIcon = (size) => {
    let iconSize, iconAnchor;

    switch (size) {
      case 'small':
```

```
        iconSize = [20, 30];
        iconAnchor = [10, 30];
        break;
      case 'large':
        iconSize = [40, 60];
        iconAnchor = [20, 60];
        break;
      case 'medium':
      default:
        iconSize = [30, 45];
        iconAnchor = [15, 45];
    }

    return L.icon({
      iconUrl: 'https://unpkg.com/leaflet@1.7.1/dist/images/marker-icon.png',
      shadowUrl:
        'https://unpkg.com/leaflet@1.7.1/dist/images/marker-shadow.png',
      iconSize: iconSize,
      iconAnchor: iconAnchor,
      popupAnchor: [0, -iconSize[1]],
      shadowSize: [iconSize[0] + 20, iconSize[1]],
    });
  };

  // Handle touch simulation
  const handleMapTouch = (e) => {
    if (!touchSimulation) return;

    // Get touch coordinates
    const touchLatlng = e.latlng;

    setTouchPoints((prev) => [
      ...prev,
      {
        id: Date.now(),
        position: [touchLatlng.lat, touchLatlng.lng],
      },
    ]);
  };

  // Clear touch points
  const clearTouchPoints = () => {
    setTouchPoints([]);
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Mobile Responsiveness & Touch Interactions</h3>
        {deviceInfo ? (
          <div>
            <p>
              <strong>Device Type:</strong> {deviceInfo.deviceType}
              <br />
```

```
            <strong>Touch Support:</strong>{' '}
            {deviceInfo.isTouchDevice ? 'Yes' : 'No'}
            <br />
            <strong>Screen Size:</strong> {deviceInfo.screenWidth}x
            {deviceInfo.screenHeight}px
            <br />
            <strong>Map Size:</strong> {deviceInfo.mapSize.x}x
            {deviceInfo.mapSize.y}px
          </p>
        </div>
      ) : (
        <p>Detecting device information...</p>
      )}

      <div
        style={{
          display: 'flex',
          flexWrap: 'wrap',
          gap: '20px',
          marginBottom: '10px',
        }}
      >
        <div>
          <label>Marker Size: </label>
          <select
            value={markerSize}
            onChange={(e) => setMarkerSize(e.target.value)}
          >
            <option value="small">Small (for desktop)</option>
            <option value="medium">Medium (balanced)</option>
            <option value="large">Large (for touch)</option>
          </select>
        </div>

        <div>
          <label>Control Position: </label>
          <select
            value={controlPosition}
            onChange={(e) => setControlPosition(e.target.value)}
          >
            <option value="topleft">Top Left</option>
            <option value="topright">Top Right</option>
            <option value="bottomleft">Bottom Left</option>
            <option value="bottomright">Bottom Right</option>
          </select>
        </div>

        <div>
          <label>
            <input
              type="checkbox"
              checked={touchSimulation}
              onChange={(e) => setTouchSimulation(e.target.checked)}
            />
```

```
              Touch Point Simulation
            </label>
            {touchSimulation && (
              <button onClick={clearTouchPoints} style={{ marginLeft: '10px' }}>
                Clear Points
              </button>
            )}
          </div>
        </div>
      </div>

      <div
        style={{
          width: '100%',
          height: deviceInfo?.deviceType === 'mobile' ? '350px' : '500px',
          position: 'relative',
        }}
      >
        <MapContainer
          center={[51.505, -0.11]}
          zoom={13}
          style={{ height: '100%', width: '100%' }}
          zoomControl={false} // We'll add ZoomControl separately to position it
          tap={true} // Enable tap for touch devices
          touchZoom={true}
          doubleClickZoom={true}
          eventHandlers={{
            click: handleMapTouch,
          }}
        >
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />

          <ZoomControl position={controlPosition} />

          <DeviceDetector onDeviceInfoUpdate={setDeviceInfo} />

          {/* Points of interest */}
          {pois.map((poi) => (
            <Marker
              key={poi.id}
              position={poi.position}
              icon={createMarkerIcon(markerSize)}
            >
              <Popup>
                <div
                  style={{
                    fontSize:
                      deviceInfo?.deviceType === 'mobile' ? '14px' : '16px',
                    padding:
                      deviceInfo?.deviceType === 'mobile' ? '5px' : '10px',
```

```
                }}
              >
                <h3 style={{ margin: '0 0 5px 0' }}>{poi.name}</h3>
                <p style={{ margin: '0' }}>{poi.description}</p>
              </div>
            </Popup>
          </Marker>
        ))}

        {/* Touch simulation points */}
        {touchPoints.map((point) => (
          <CircleMarker
            key={point.id}
            center={point.position}
            radius={10}
            pathOptions={{
              color: '#ff4081',
              fillColor: '#ff4081',
              fillOpacity: 0.5,
            }}
          >
            <Popup>
              Touch at {point.position[0].toFixed(5)},{' '}
              {point.position[1].toFixed(5)}
            </Popup>
          </CircleMarker>
        ))}
      </MapContainer>
    </div>

    <div style={{ marginTop: '15px' }}>
      <h4>Mobile & Touch Design Considerations</h4>
      <ul>
        <li>
          <strong>Larger Touch Targets</strong>: Increase marker size and
          control buttons for finger tapping
        </li>
        <li>
          <strong>Control Positioning</strong>: Place controls where they
          won't be obscured by thumbs (often bottom-left)
        </li>
        <li>
          <strong>Reduced Information Density</strong>: Show fewer details on
          small screens
        </li>
        <li>
          <strong>Responsive Popups</strong>: Adjust popup size and content
          based on screen size
        </li>
        <li>
          <strong>Custom Touch Gestures</strong>: Consider implementing
          pinch-to-zoom and custom swipe actions
        </li>
        <li>
```

```
            <strong>Performance Optimization</strong>: Mobile devices often have
            less processing power and memory
          </li>
        </ul>

        <h4>Implementation Tips</h4>
        <ul>
          <li>
            Set <code>tap: true</code> on MapContainer to optimize for touch
            devices
          </li>
          <li>
            Use media queries in CSS to adjust map container size based on
            viewport width
          </li>
          <li>Test on actual devices, not just browser emulators</li>
          <li>Consider device orientation changes (portrait vs landscape)</li>
          <li>
            Implement responsive markers with different sizes based on screen
            width
          </li>
          <li>
            Optimize for mobile networks by reducing data usage where possible
          </li>
        </ul>
      </div>
    </div>
  );
};

export default MobileResponsiveExample;
```

Accessibility Considerations

```
import React, { useState, useEffect, useRef } from 'react';
import { MapContainer, TileLayer, Marker, Popup, useMap } from 'react-leaflet';
import L from 'leaflet';

// Component to add keyboard navigation
const KeyboardNavigation = () => {
  const map = useMap();
  const [focusedMarkerIndex, setFocusedMarkerIndex] = useState(-1);

  // Sample points of interest
  const pois = [
    {
      id: 1,
      name: 'London Eye',
      position: [51.503, -0.119],
      description: 'Giant Ferris wheel',
    },
```

```
    {
      id: 2,
      name: 'Tower of London',
      position: [51.508, -0.076],
      description: 'Historic castle',
    },
    {
      id: 3,
      name: 'Hyde Park',
      position: [51.507, -0.165],
      description: 'Major park',
    },
    {
      id: 4,
      name: 'British Museum',
      position: [51.519, -0.127],
      description: 'Public museum',
    },
    {
      id: 5,
      name: 'Buckingham Palace',
      position: [51.501, -0.142],
      description: 'Royal residence',
    },
  ];

  // Reference to marker elements
  const markerRefs = useRef([]);

  // Set up keyboard handlers
  useEffect(() => {
    // Prepare map container for keyboard interaction
    const mapContainer = map.getContainer();

    if (!mapContainer.hasAttribute('tabindex')) {
      mapContainer.setAttribute('tabindex', '0');
    }

    mapContainer.setAttribute('role', 'application');
    mapContainer.setAttribute(
      'aria-label',
      'Interactive map of London landmarks'
    );

    // Function to handle keyboard navigation
    const handleKeyDown = (e) => {
      // Only handle events when map container is focused
      if (!mapContainer.contains(document.activeElement)) return;

      switch (e.key) {
        case 'Tab':
          // Tab behavior is handled by the browser
          break;
```

```
        case 'ArrowRight':
        case 'ArrowDown':
          // Navigate to next marker
          e.preventDefault();
          setFocusedMarkerIndex((prev) => {
            const newIndex = prev < pois.length - 1 ? prev + 1 : 0;

            // Update map view and open popup
            if (newIndex >= 0) {
              map.setView(pois[newIndex].position, map.getZoom());
              markerRefs.current[newIndex]?.openPopup();
            }

            return newIndex;
          });
          break;

        case 'ArrowLeft':
        case 'ArrowUp':
          // Navigate to previous marker
          e.preventDefault();
          setFocusedMarkerIndex((prev) => {
            const newIndex = prev > 0 ? prev - 1 : pois.length - 1;

            // Update map view and open popup
            if (newIndex >= 0) {
              map.setView(pois[newIndex].position, map.getZoom());
              markerRefs.current[newIndex]?.openPopup();
            }

            return newIndex;
          });
          break;

        case 'Escape':
          // Clear focus
          setFocusedMarkerIndex(-1);
          map.closePopup();
          break;

        case '+':
        case '=':
          // Zoom in
          e.preventDefault();
          map.zoomIn();
          break;

        case '-':
        case '_':
          // Zoom out
          e.preventDefault();
          map.zoomOut();
          break;
```

```
          case 'Home':
            // Reset view
            e.preventDefault();
            map.setView([51.505, -0.11], 13);
            break;

          default:
            // Other keys
            break;
        }
      };

      // Add event listener
      mapContainer.addEventListener('keydown', handleKeyDown);

      // Add instructions
      const instructionsEl = document.createElement('div');
      instructionsEl.setAttribute('role', 'region');
      instructionsEl.setAttribute('aria-label', 'Map keyboard controls');
      instructionsEl.setAttribute('id', 'map-instructions');
      instructionsEl.style.position = 'absolute';
      instructionsEl.style.left = '-9999px';
      instructionsEl.style.top = 'auto';
      instructionsEl.style.width = '1px';
      instructionsEl.style.height = '1px';
      instructionsEl.style.overflow = 'hidden';

      instructionsEl.innerHTML = `
        <p>Use arrow keys to navigate between landmarks. Press Plus to zoom in,
Minus to zoom out, Home to reset view.</p>
      `;

      mapContainer.prepend(instructionsEl);
      mapContainer.setAttribute('aria-describedby', 'map-instructions');

      // Clean up
      return () => {
        mapContainer.removeEventListener('keydown', handleKeyDown);
        instructionsEl.remove();
      };
    }, [map]);

    return (
      <>
        {pois.map((poi, index) => (
          <Marker
            key={poi.id}
            position={poi.position}
            ref={(element) => (markerRefs.current[index] = element)}
            eventHandlers={{
              click: () => {
                setFocusedMarkerIndex(index);
              },
            }}
```

```
          >
            <Popup>
              <div>
                <h3 id={`poi-${poi.id}-heading`}>{poi.name}</h3>
                <p id={`poi-${poi.id}-description`}>{poi.description}</p>
              </div>
            </Popup>
          </Marker>
        ))}
      </>
    );
  };

  // Component for screen reader announcements
  const AccessibilityExample = () => {
    const [announcement, setAnnouncement] = useState('');
    const [highContrastMode, setHighContrastMode] = useState(false);
    const [largeText, setLargeText] = useState(false);
    const [focusIndicators, setFocusIndicators] = useState(true);

    // Simulate an accessible alternative
    const [showTextAlternative, setShowTextAlternative] = useState(false);

    // Sample points of interest for text alternative
    const pois = [
      {
        id: 1,
        name: 'London Eye',
        position: [51.503, -0.119],
        description: 'Giant Ferris wheel on the South Bank of the River Thames',
      },
      {
        id: 2,
        name: 'Tower of London',
        position: [51.508, -0.076],
        description: 'Historic castle on the north bank of the River Thames',
      },
      {
        id: 3,
        name: 'Hyde Park',
        position: [51.507, -0.165],
        description: 'Major park in central London',
      },
      {
        id: 4,
        name: 'British Museum',
        position: [51.519, -0.127],
        description: 'Public museum dedicated to human history, art and culture',
      },
      {
        id: 5,
        name: 'Buckingham Palace',
        position: [51.501, -0.142],
        description:
```

```
        'London residence and administrative headquarters of the monarch of the
United Kingdom',
    },
  ];

  // Define CSS classes based on accessibility settings
  const getMapClasses = () => {
    let classes = '';
    if (highContrastMode) classes += 'high-contrast-map ';
    if (largeText) classes += 'large-text-map ';
    if (focusIndicators) classes += 'focus-indicators-map';
    return classes.trim();
  };

  // Custom CSS for accessibility settings
  const accessibilityStyles = `
    .high-contrast-map .leaflet-tile {
      filter: contrast(1.4) saturate(0.8) brightness(1.1);
    }

    .high-contrast-map .leaflet-marker-icon {
      border: 2px solid black;
      box-shadow: 0 0 0 2px white;
    }

    .large-text-map .leaflet-popup-content {
      font-size: 1.2em;
      line-height: 1.5;
    }

    .large-text-map .leaflet-control {
      font-size: 1.2em;
    }

    .focus-indicators-map .leaflet-interactive:focus {
      outline: 4px solid orange !important;
      outline-offset: 2px;
    }
  `;

  // Function to make announcement for screen readers
  const announce = (message) => {
    setAnnouncement(message);
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Accessibility Considerations for Maps</h3>
        <p>
          Maps are inherently visual, but we can make them more accessible
          through several techniques.
        </p>
```

```
          <div
            style={{
              display: 'flex',
              flexWrap: 'wrap',
              gap: '15px',
              marginBottom: '15px',
            }}
          >
            <div>
              <label>
                <input
                  type="checkbox"
                  checked={highContrastMode}
                  onChange={(e) => setHighContrastMode(e.target.checked)}
                />
                High Contrast Mode
              </label>
            </div>

            <div>
              <label>
                <input
                  type="checkbox"
                  checked={largeText}
                  onChange={(e) => setLargeText(e.target.checked)}
                />
                Large Text
              </label>
            </div>

            <div>
              <label>
                <input
                  type="checkbox"
                  checked={focusIndicators}
                  onChange={(e) => setFocusIndicators(e.target.checked)}
                />
                Focus Indicators
              </label>
            </div>

            <div>
              <label>
                <input
                  type="checkbox"
                  checked={showTextAlternative}
                  onChange={(e) => setShowTextAlternative(e.target.checked)}
                />
                Show Text Alternative
              </label>
            </div>
          </div>

          <div style={{ marginBottom: '15px' }}>
```

```
            <button
              onClick={() =>
                announce(
                  'Map showing 5 landmarks in London including London Eye, Tower of
London, and Buckingham Palace.'
                )
              }
              style={{ marginRight: '10px' }}
            >
              Announce Map Content
            </button>

            <button
              onClick={() =>
                announce(
                  'Zoom level increased to 14. Now showing central London area.'
                )
              }
              style={{ marginRight: '10px' }}
            >
              Simulate Zoom Announcement
            </button>

            <button
              onClick={() =>
                announce(
                  'Selected landmark: Buckingham Palace, royal residence in
Westminster.'
                )
              }
            >
              Simulate Selection Announcement
            </button>
          </div>
        </div>

        {/* Screen reader announcer */}
        <Announcer message={announcement} />

        {/* Inject accessibility styles */}
        <style dangerouslySetInnerHTML={{ __html: accessibilityStyles }} />

        {showTextAlternative ? (
          <div
            className={largeText ? 'large-text-map' : ''}
            style={{
              padding: '15px',
              border: '1px solid #ddd',
              borderRadius: '4px',
              backgroundColor: highContrastMode ? '#000' : '#f9f9f9',
              color: highContrastMode ? '#fff' : '#333',
            }}
          >
            <h3>Text Alternative: London Landmarks Map</h3>
```

```
            <p>This map shows 5 major landmarks in central London:</p>
            <ol>
              {pois.map((poi) => (
                <li key={poi.id} style={{ marginBottom: '10px' }}>
                  <strong>{poi.name}</strong> - Located at coordinates{' '}
                  {poi.position[0].toFixed(4)}, {poi.position[1].toFixed(4)}.{' '}
                  {poi.description}
                </li>
              ))}
            </ol>
            <p>
              The landmarks are distributed across central London, with Hyde Park
              being the westernmost point and Tower of London being the
              easternmost point.
            </p>
          </div>
        ) : (
          <div
            className={getMapClasses()}
            style={{ height: '500px', width: '100%', position: 'relative' }}
          >
            <MapContainer
              center={[51.505, -0.11]}
              zoom={13}
              style={{ height: '100%', width: '100%' }}
              // Improve keyboard navigation
              whenCreated={(map) => {
                const container = map.getContainer();
                container.setAttribute(
                  'aria-label',
                  'Interactive map of London landmarks'
                );
                container.setAttribute('role', 'application');
                container.setAttribute('tabindex', '0');
              }}
            >
              <TileLayer
                url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
                attribution='&copy; <a
  href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
              />

              <KeyboardNavigation />
            </MapContainer>

            {/* Skip link for keyboard users */}
            <a
              href="#map-alternative-content"
              style={{
                position: 'absolute',
                left: '0',
                top: '-40px',
                padding: '8px',
                backgroundColor: '#333',
```

```
              color: '#fff',
              zIndex: 1000,
              textDecoration: 'none',
              transition: 'top 0.3s',
              ':focus': {
                top: '0',
              },
            }}
            onFocus={(e) => {
              e.target.style.top = '0';
            }}
            onBlur={(e) => {
              e.target.style.top = '-40px';
            }}
          >
            Skip map navigation
          </a>
        </div>
      )}

      <div id="map-alternative-content" style={{ marginTop: '15px' }}>
        <h4>WCAG Compliance Considerations</h4>
        <ul>
          <li>
            <strong>Perceivable</strong>: Provide text alternatives, ensure
            sufficient contrast, and allow content to be presented in different
            ways
          </li>
          <li>
            <strong>Operable</strong>: Ensure map can be navigated with
            keyboard, provide skip links, and avoid content that could cause
            seizures
          </li>
          <li>
            <strong>Understandable</strong>: Make content readable, predictable,
            and help users avoid and correct mistakes
          </li>
          <li>
            <strong>Robust</strong>: Ensure compatibility with current and
            future assistive technologies
          </li>
        </ul>

        <h4>Accessibility Implementation Checklist</h4>
        <ul>
          <li>
            Provide text alternatives for maps (descriptions, data tables, etc.)
          </li>
          <li>
            Add proper keyboard navigation support (focus management, keyboard
            shortcuts)
          </li>
          <li>
            Ensure proper color contrast for map controls and informational
```

```
            layers
          </li>
          <li>Add screen reader announcements for important map events</li>
          <li>Implement "skip map" links for keyboard users</li>
          <li>Ensure all popups and information windows are accessible</li>
          <li>Add appropriate ARIA attributes to map elements</li>
          <li>
            Test with actual assistive technologies (screen readers,
            keyboard-only navigation)
          </li>
          <li>
            Add options for high contrast mode, larger text, and simplified
            views
          </li>
        </ul>
      </div>
    </div>
  );
};

export default AccessibilityExample;
```

Performance Optimization Techniques

Optimizing map performance is crucial, especially for applications with large datasets or complex interactions:

```
import React, { useState, useEffect, useRef } from 'react';
import { MapContainer, TileLayer, LayersControl, useMap } from 'react-leaflet';
import L from 'leaflet';

// Component for performance monitoring
const PerformanceMonitor = ({ onStatsUpdate }) => {
  const map = useMap();
  const frameCountRef = useRef(0);
  const lastTimeRef = useRef(performance.now());
  const fpsRef = useRef(0);

  // Track render time for map movements
  useEffect(() => {
    const updateStats = () => {
      const now = performance.now();
      const elapsed = now - lastTimeRef.current;

      if (elapsed >= 1000) {
        // Update every second
        fpsRef.current = Math.round((frameCountRef.current * 1000) / elapsed);

        onStatsUpdate({
          fps: fpsRef.current,
          tileCount: document.querySelectorAll('.leaflet-tile').length,
          memoryEstimate:
```

```
            Math.round(performance.memory?.usedJSHeapSize / (1024 * 1024)) ||
            'N/A',
        });

        frameCountRef.current = 0;
        lastTimeRef.current = now;
      }

      frameCountRef.current++;
      requestAnimationFrame(updateStats);
    };

    // Start monitoring
    const animationId = requestAnimationFrame(updateStats);

    // Monitor map events
    const handleMoveStart = () => {
      onStatsUpdate((prev) => ({
        ...prev,
        lastEvent: 'movestart',
        eventTime: performance.now(),
      }));
    };

    const handleMoveEnd = () => {
      const endTime = performance.now();
      const startTime = onStatsUpdate((prev) => prev.eventTime || endTime);
      const duration = endTime - startTime;

      onStatsUpdate((prev) => ({
        ...prev,
        lastEvent: 'moveend',
        lastMoveDuration: duration.toFixed(2),
      }));
    };

    const handleZoomStart = () => {
      onStatsUpdate((prev) => ({
        ...prev,
        lastEvent: 'zoomstart',
        eventTime: performance.now(),
      }));
    };

    const handleZoomEnd = () => {
      const endTime = performance.now();
      const startTime = onStatsUpdate((prev) => prev.eventTime || endTime);
      const duration = endTime - startTime;

      onStatsUpdate((prev) => ({
        ...prev,
        lastEvent: 'zoomend',
        lastZoomDuration: duration.toFixed(2),
      }));
```

```
    };

    map.on('movestart', handleMoveStart);
    map.on('moveend', handleMoveEnd);
    map.on('zoomstart', handleZoomStart);
    map.on('zoomend', handleZoomEnd);

    return () => {
      cancelAnimationFrame(animationId);
      map.off('movestart', handleMoveStart);
      map.off('moveend', handleMoveEnd);
      map.off('zoomstart', handleZoomStart);
      map.off('zoomend', handleZoomEnd);
    };
  }, [map, onStatsUpdate]);

  return null;
};

// Debounce utility function
const debounce = (func, wait) => {
  let timeout;
  return function executedFunction(...args) {
    const later = () => {
      clearTimeout(timeout);
      func(...args);
    };
    clearTimeout(timeout);
    timeout = setTimeout(later, wait);
  };
};

const PerformanceOptimizationExample = () => {
  const [stats, setStats] = useState({
    fps: 0,
    tileCount: 0,
    memoryEstimate: 'N/A',
    lastEvent: 'none',
    lastMoveDuration: 0,
    lastZoomDuration: 0,
  });

  const [optimizations, setOptimizations] = useState({
    tileBuffering: true,
    simplifyGeometry: true,
    debounceUpdates: true,
    limitFPS: false,
    lazyLoading: true,
  });

  const [simpleMode, setSimpleMode] = useState(false);
  const [stressTest, setStressTest] = useState(false);
  const mapRef = useRef(null);
  const shapesLayerRef = useRef(null);
```

```
// Toggle an optimization
const toggleOptimization = (key) => {
  setOptimizations((prev) => ({
    ...prev,
    [key]: !prev[key],
  }));
};

// Create a stress test layer with many shapes
const createStressTestLayer = () => {
  if (!mapRef.current) return;

  // Clear existing layer
  if (shapesLayerRef.current) {
    shapesLayerRef.current.clearLayers();
  } else {
    shapesLayerRef.current = L.layerGroup().addTo(mapRef.current);
  }

  const map = mapRef.current;
  const bounds = map.getBounds();
  const southWest = bounds.getSouthWest();
  const northEast = bounds.getNorthEast();

  const latRange = northEast.lat - southWest.lat;
  const lngRange = northEast.lng - southWest.lng;

  // Determine shape count based on optimization settings
  const shapeCount = simpleMode ? 100 : 500;

  // Create shapes with different complexities based on optimization settings
  for (let i = 0; i < shapeCount; i++) {
    const shapeType = Math.random() > 0.5 ? 'circle' : 'polygon';

    if (shapeType === 'circle') {
      const lat = southWest.lat + Math.random() * latRange;
      const lng = southWest.lng + Math.random() * lngRange;
      const radius = Math.random() * 500 + 100;

      L.circle([lat, lng], {
        radius,
        color: getRandomColor(),
        weight: 2,
        opacity: 0.7,
        fillOpacity: 0.3,
      }).addTo(shapesLayerRef.current);
    } else {
      const centerLat = southWest.lat + Math.random() * latRange;
      const centerLng = southWest.lng + Math.random() * lngRange;

      // Number of points in polygon
      const vertexCount = optimizations.simplifyGeometry ? 8 : 20;
```

```
        const points = [];
        const radius = Math.random() * 0.01 + 0.003;

        for (let j = 0; j < vertexCount; j++) {
          const angle = (j / vertexCount) * Math.PI * 2;
          const lat = centerLat + Math.cos(angle) * radius;
          const lng = centerLng + Math.sin(angle) * radius;
          points.push([lat, lng]);
        }

        // Close the polygon
        points.push(points[0]);

        L.polygon(points, {
          color: getRandomColor(),
          weight: 2,
          opacity: 0.7,
          fillOpacity: 0.3,
        }).addTo(shapesLayerRef.current);
      }
    }
  };

  // Get a random color
  const getRandomColor = () => {
    const letters = '0123456789ABCDEF';
    let color = '#';
    for (let i = 0; i < 6; i++) {
      color += letters[Math.floor(Math.random() * 16)];
    }
    return color;
  };

  // Apply optimization settings to the map
  useEffect(() => {
    if (!mapRef.current) return;

    const map = mapRef.current;

    // Apply tile buffering optimization
    map.options.maxBoundsViscosity = optimizations.tileBuffering ? 1.0 : 0;

    // Apply FPS limiting (for demo purposes)
    if (optimizations.limitFPS) {
      if (!map._originalInvalidateSize) {
        map._originalInvalidateSize = map.invalidateSize;
        map.invalidateSize = debounce(
          map._originalInvalidateSize.bind(map),
          100
        );
      }
    } else if (map._originalInvalidateSize) {
      map.invalidateSize = map._originalInvalidateSize;
      delete map._originalInvalidateSize;
```

```
    }

    // Apply lazy loading for stress test
    if (stressTest) {
      if (optimizations.debounceUpdates) {
        createStressTestLayer = debounce(createStressTestLayer, 300);
      }

      createStressTestLayer();
    } else if (shapesLayerRef.current) {
      shapesLayerRef.current.clearLayers();
    }
  }, [optimizations, stressTest, simpleMode]);

  // Handle map instance creation
  const handleMapCreated = (map) => {
    mapRef.current = map;

    // Apply initial optimization settings
    map.options.preferCanvas = true;
    map.options.zoomSnap = 0.5;
    map.options.zoomDelta = 0.5;
    map.options.wheelPxPerZoomLevel = 120;

    if (optimizations.tileBuffering) {
      // Buffer more tiles outside viewport
      map.options.maxBoundsViscosity = 1.0;
    }
  };

  return (
    <div>
      <div style={{ marginBottom: '10px' }}>
        <h3>Performance Optimization Techniques</h3>
        <p>
          Optimize your map for better performance, especially when dealing with
          complex data or mobile devices.
        </p>

        <div
          style={{
            display: 'flex',
            flexWrap: 'wrap',
            gap: '15px',
            marginBottom: '15px',
            padding: '10px',
            backgroundColor: '#f5f5f5',
            borderRadius: '4px',
          }}
        >
          <div>
            <h4 style={{ margin: '0 0 8px 0' }}>Performance Stats</h4>
            <div>
              <strong>FPS:</strong> {stats.fps}
```

```
            <br />
            <strong>Tile Count:</strong> {stats.tileCount}
            <br />
            <strong>Last Event:</strong> {stats.lastEvent}
            <br />
            {stats.lastMoveDuration > 0 && (
              <div>
                <strong>Last Move Duration:</strong> {stats.lastMoveDuration}
                ms
              </div>
            )}
            {stats.lastZoomDuration > 0 && (
              <div>
                <strong>Last Zoom Duration:</strong> {stats.lastZoomDuration}
                ms
              </div>
            )}
            <strong>Memory Use:</strong> {stats.memoryEstimate} MB
          </div>
        </div>

        <div>
          <h4 style={{ margin: '0 0 8px 0' }}>Optimizations</h4>
          <div>
            <label>
              <input
                type="checkbox"
                checked={optimizations.tileBuffering}
                onChange={() => toggleOptimization('tileBuffering')}
              />
              Tile Buffering
            </label>
            <br />

            <label>
              <input
                type="checkbox"
                checked={optimizations.simplifyGeometry}
                onChange={() => toggleOptimization('simplifyGeometry')}
              />
              Simplify Geometry
            </label>
            <br />

            <label>
              <input
                type="checkbox"
                checked={optimizations.debounceUpdates}
                onChange={() => toggleOptimization('debounceUpdates')}
              />
              Debounce Updates
            </label>
            <br />
```

```
              <label>
                <input
                  type="checkbox"
                  checked={optimizations.limitFPS}
                  onChange={() => toggleOptimization('limitFPS')}
                />
                Limit FPS
              </label>
              <br />

              <label>
                <input
                  type="checkbox"
                  checked={optimizations.lazyLoading}
                  onChange={() => toggleOptimization('lazyLoading')}
                />
                Lazy Loading
              </label>
            </div>
          </div>

          <div>
            <h4 style={{ margin: '0 0 8px 0' }}>Test Options</h4>
            <div>
              <label>
                <input
                  type="checkbox"
                  checked={simpleMode}
                  onChange={() => setSimpleMode(!simpleMode)}
                />
                Simple Mode (Fewer Elements)
              </label>
              <br />

              <label>
                <input
                  type="checkbox"
                  checked={stressTest}
                  onChange={() => setStressTest(!stressTest)}
                />
                Stress Test (Many Elements)
              </label>
            </div>
          </div>
        </div>
      </div>

      <MapContainer
        center={[51.505, -0.09]}
        zoom={13}
        style={{ height: '500px', width: '100%' }}
        preferCanvas={true}
        whenCreated={handleMapCreated}
      >
```

```
          <TileLayer
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
          />

          <PerformanceMonitor onStatsUpdate={setStats} />
        </MapContainer>

        <div style={{ marginTop: '15px' }}>
          <h4>Performance Optimization Techniques</h4>

          <div style={{ marginBottom: '10px' }}>
            <strong>Rendering Optimizations</strong>
            <ul>
              <li>
                <strong>Use Canvas Rendering</strong>: Set{' '}
                <code>preferCanvas={true}</code> for better performance with many
                vector layers
              </li>
              <li>
                <strong>Simplify Complex Geometries</strong>: Reduce the number of
                points in polygons and lines
              </li>
              <li>
                <strong>Layer Management</strong>: Only show necessary layers at
                each zoom level
              </li>
              <li>
                <strong>Clustering</strong>: Group nearby markers to reduce DOM
                elements
              </li>
              <li>
                <strong>Virtual Rendering</strong>: Only render elements in or
                near the viewport
              </li>
            </ul>
          </div>

          <div style={{ marginBottom: '10px' }}>
            <strong>Interaction Optimizations</strong>
            <ul>
              <li>
                <strong>Debounce Event Handlers</strong>: Limit the frequency of
                expensive operations
              </li>
              <li>
                <strong>Throttle Map Updates</strong>: Reduce updates during
                continuous interactions like panning
              </li>
              <li>
                <strong>Lazy Loading</strong>: Load data progressively as needed
              </li>
              <li>
```

```
            <strong>Preloading</strong>: Load adjacent data before it's needed
          </li>
          <li>
            <strong>Caching</strong>: Store processed data to avoid
            recalculation
          </li>
        </ul>
      </div>

      <div style={{ marginBottom: '10px' }}>
        <strong>Leaflet-Specific Optimizations</strong>
        <ul>
          <li>
            <strong>setZoomSnap and setZoomDelta</strong>: Adjust for smoother
            zooming
          </li>
          <li>
            <strong>maxBounds and maxBoundsViscosity</strong>: Limit map area
            and tile loading
          </li>
          <li>
            <strong>Custom Panes</strong>: Control rendering order and
            optimize layer management
          </li>
          <li>
            <strong>Renderer Options</strong>: Tune vector rendering
            parameters
          </li>
          <li>
            <strong>Transform Animations</strong>: Use CSS transforms instead
            of forced reflows
          </li>
        </ul>
      </div>

      <div>
        <strong>React-Specific Optimizations</strong>
        <ul>
          <li>
            <strong>Memoization</strong>: Use React.memo, useMemo, and
            useCallback to prevent unnecessary renders
          </li>
          <li>
            <strong>State Management</strong>: Organize state to minimize
            re-renders and updates
          </li>
          <li>
            <strong>Component Structure</strong>: Separate map components to
            isolate re-renders
          </li>
          <li>
            <strong>Event Handling</strong>: Use useMap hook instead of
            passing map through props
          </li>
```

```
            <li>
              <strong>Refs</strong>: Use refs for imperative operations instead
              of state when appropriate
            </li>
          </ul>
        </div>
      </div>
    </div>
  );
};

export default PerformanceOptimizationExample;
```

Testing Strategies for Map Components

Testing map components requires special approaches due to their visual nature and dependence on external services:

```
import React from 'react';

const TestingStrategiesGuide = () => {
  return (
    <div>
      <h3>Testing Strategies for Map Components</h3>

      <div style={{ marginBottom: '20px' }}>
        <p>
          Testing interactive map components presents unique challenges. Here
          are comprehensive strategies for different types of testing.
        </p>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Unit Testing Map Components</h4>

        <div style={{ marginBottom: '15px' }}>
          <h5>Testing Setup with Jest and React Testing Library</h5>
          <pre
            style={{
              backgroundColor: '#f5f5f5',
              padding: '15px',
              borderRadius: '4px',
              overflowX: 'auto',
            }}
          >
            {`// Import testing utilities
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import '@testing-library/jest-dom';
import MapComponent from './MapComponent';

// Mock Leaflet modules
```

```
jest.mock('react-leaflet', () => ({
  MapContainer: jest.fn(({ children }) => <div data-testid="map-container">
{children}</div>),
  TileLayer: jest.fn(() => <div data-testid="tile-layer" />),
  Marker: jest.fn(({ children }) => <div data-testid="marker">{children}</div>),
  Popup: jest.fn(({ children }) => <div data-testid="popup">{children}</div>),
  useMap: jest.fn(() => ({
    setView: jest.fn(),
    flyTo: jest.fn(),
    getZoom: jest.fn(() => 13),
    on: jest.fn(),
    off: jest.fn()
  }))
}));

// Mock Leaflet
jest.mock('leaflet', () => ({
  icon: jest.fn(() => ({})),
  latLng: jest.fn((lat, lng) => ({ lat, lng })),
  layerGroup: jest.fn(() => ({
    addTo: jest.fn(() => ({})),
    clearLayers: jest.fn()
  }))
}));

// Setup mock for global L object
global.L = {
  latLng: jest.fn((lat, lng) => ({ lat, lng }))
};

describe('MapComponent', () => {
  test('renders map with initial markers', () => {
    const initialMarkers = [
      { id: 1, position: [51.505, -0.09], name: 'Location 1' },
      { id: 2, position: [51.51, -0.1], name: 'Location 2' }
    ];

    render(<MapComponent markers={initialMarkers} />);

    // Check that map container is rendered
    expect(screen.getByTestId('map-container')).toBeInTheDocument();

    // Check that markers are rendered
    const markers = screen.getAllByTestId('marker');
    expect(markers).toHaveLength(2);

    // Check marker content
    expect(screen.getByText('Location 1')).toBeInTheDocument();
    expect(screen.getByText('Location 2')).toBeInTheDocument();
  });

  test('adds new marker when add marker button is clicked', async () => {
    render(<MapComponent />);
```

```
    // Initial state - no markers
    expect(screen.queryAllByTestId('marker')).toHaveLength(0);

    // Click add marker button
    fireEvent.click(screen.getByText('Add Marker'));

    // Wait for marker to be added
    await waitFor(() => {
      expect(screen.getAllByTestId('marker')).toHaveLength(1);
    });
  });
});`}
          </pre>
        </div>

        <div>
          <h5>What to Test in Map Components</h5>
          <ul>
            <li>
              <strong>Component Rendering</strong>: Verify that the map and its
              elements render correctly
            </li>
            <li>
              <strong>Props Handling</strong>: Test that the component responds
              correctly to prop changes
            </li>
            <li>
              <strong>User Interactions</strong>: Test event handlers for
              clicks, hovers, etc.
            </li>
            <li>
              <strong>State Management</strong>: Verify that state is updated
              correctly
            </li>
            <li>
              <strong>Callbacks</strong>: Test that callback functions are
              called with correct arguments
            </li>
            <li>
              <strong>Custom Logic</strong>: Test any custom functions or
              calculations
            </li>
          </ul>
        </div>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Integration Testing</h4>

        <div style={{ marginBottom: '15px' }}>
          <h5>Using Cypress for Map Testing</h5>
          <pre
            style={{
              backgroundColor: '#f5f5f5',
```

```
              padding: '15px',
              borderRadius: '4px',
              overflowX: 'auto',
            }}
          >
            {`// cypress/integration/map.spec.js
describe('Map Component Integration', () => {
  beforeEach(() => {
    // Visit the page with the map
    cy.visit('/map-page');

    // Wait for map to load (look for a specific element)
    cy.get('.leaflet-container', { timeout: 10000 }).should('be.visible');
  });

  it('displays map with correct initial view', () => {
    // Check for map container
    cy.get('.leaflet-container')
      .should('exist')
      .and('be.visible');

    // Check for expected layers
    cy.get('.leaflet-tile-pane').should('exist');
    cy.get('.leaflet-marker-pane').should('exist');
  });

  it('allows user to add markers by clicking on map', () => {
    // Count initial markers
    cy.get('.leaflet-marker-icon').then($markers => {
      const initialCount = $markers.length;

      // Click on map
      cy.get('.leaflet-container').click(300, 300);

      // Verify new marker was added
      cy.get('.leaflet-marker-icon').should('have.length', initialCount + 1);
    });
  });

  it('shows popup when marker is clicked', () => {
    // Click on a marker
    cy.get('.leaflet-marker-icon').first().click();

    // Check that popup is displayed
    cy.get('.leaflet-popup').should('be.visible');
    cy.get('.leaflet-popup-content').should('not.be.empty');
  });

  it('zooms in when zoom control is clicked', () => {
    // Get initial zoom level from map's data attribute or UI element
    cy.get('[data-testid="zoom-level"]').then($zoomLevel => {
      const initialZoom = parseInt($zoomLevel.text());

      // Click zoom in button
```

```
      cy.get('.leaflet-control-zoom-in').click();

      // Check new zoom level
      cy.get('[data-testid="zoom-level"]').should('contain', initialZoom + 1);
    });
  });
});`}
          </pre>
        </div>

        <div>
          <h5>Integration Testing Strategies</h5>
          <ul>
            <li>
              <strong>Test Real DOM Interactions</strong>: Click on map,
              markers, controls
            </li>
            <li>
              <strong>Test Map State Changes</strong>: Zoom, pan, layer
              visibility
            </li>
            <li>
              <strong>Test With Real API Calls</strong>: For GeoJSON, geocoding,
              routing, etc.
            </li>
            <li>
              <strong>Test Data Flow</strong>: From user input to map display
            </li>
            <li>
              <strong>Test Responsive Behavior</strong>: Across different
              viewport sizes
            </li>
          </ul>
        </div>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Visual Regression Testing</h4>

        <div style={{ marginBottom: '15px' }}>
          <p>
            Visual regression testing is particularly important for maps, which
            are highly visual components. Tools like Cypress's Percy integration
            or Storybook's visual testing can help.
          </p>

          <h5>Example: Storybook with Percy Integration</h5>
          <pre
            style={{
              backgroundColor: '#f5f5f5',
              padding: '15px',
              borderRadius: '4px',
              overflowX: 'auto',
            }}
```

```
          >
            {`// MapComponent.stories.js
import React from 'react';
import { MapComponent } from './MapComponent';

export default {
  title: 'Components/MapComponent',
  component: MapComponent,
  parameters: {
    // Add viewports for responsive testing
    viewport: {
      viewports: {
        mobile: { width: 375, height: 667 },
        tablet: { width: 768, height: 1024 },
        desktop: { width: 1280, height: 800 },
      },
      defaultViewport: 'desktop',
    },
  },
};

// Basic map
export const Default = () => <MapComponent />;

// Map with markers
export const WithMarkers = () => (
  <MapComponent
    markers={[
      { id: 1, position: [51.505, -0.09], name: 'Marker 1' },
      { id: 2, position: [51.51, -0.1], name: 'Marker 2' }
    ]}
  />
);

// Map with different tile layer
export const SatelliteView = () => (
  <MapComponent
    tileLayer={{
      url:
'https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile
/{z}/{y}/{x}',
      attribution: 'Tiles &copy; Esri'
    }}
  />
);

// Map with GeoJSON data
export const WithGeoJSON = () => (
  <MapComponent
    geoJsonData={sampleGeoJSON}
    center={[40.73, -73.93]}
    zoom={12}
  />
);`}
```

```
            </pre>
          </div>

          <div>
            <h5>Visual Testing Considerations</h5>
            <ul>
              <li>
                <strong>Stabilize Map State</strong>: Ensure consistent initial
                state for tests
              </li>
              <li>
                <strong>Mock External Resources</strong>: Use consistent tile
                imagery
              </li>
              <li>
                <strong>Test Across Devices</strong>: Mobile, tablet, desktop
              </li>
              <li>
                <strong>Test Animations</strong>: Transitions, popups, markers
              </li>
              <li>
                <strong>Test Accessibility Features</strong>: High contrast mode,
                focus indicators
              </li>
            </ul>
          </div>
        </div>

        <div style={{ marginBottom: '25px' }}>
          <h4>Performance Testing</h4>

          <div style={{ marginBottom: '15px' }}>
            <p>
              Performance testing is crucial for maps that may render large
              datasets or complex visualizations.
            </p>

            <h5>Example: Performance Test with Puppeteer</h5>
            <pre
              style={{
                backgroundColor: '#f5f5f5',
                padding: '15px',
                borderRadius: '4px',
                overflowX: 'auto',
              }}
            >
              {`// performance-test.js
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();

  // Enable performance metrics
```

```
  await page.setViewport({ width: 1280, height: 800 });
  await page.setCacheEnabled(false);

  // Navigate to page with map
  await page.goto('http://localhost:3000/map-with-data');

  // Wait for map to fully load
  await page.waitForSelector('.leaflet-container');

  console.log('Testing map load performance...');

  // Measure initial load performance
  const initialPerformance = await page.evaluate(() => {
    return {
      timing: window.performance.timing.toJSON(),
      memory: window.performance.memory ?
        window.performance.memory.usedJSHeapSize / (1024 * 1024) : 'N/A'
    };
  });

  console.log('Initial load memory usage (MB):', initialPerformance.memory);
  console.log('Load time (ms):',
    initialPerformance.timing.loadEventEnd -
initialPerformance.timing.navigationStart);

  // Test interaction performance (e.g., panning)
  console.log('Testing map interaction performance...');

  // Start performance recording
  await page.evaluate(() => window.performance.mark('panStart'));

  // Simulate panning
  const mapElement = await page.$('.leaflet-container');
  const mapBounds = await mapElement.boundingBox();
  const startX = mapBounds.x + mapBounds.width / 2;
  const startY = mapBounds.y + mapBounds.height / 2;

  await page.mouse.move(startX, startY);
  await page.mouse.down();
  await page.mouse.move(startX - 200, startY, { steps: 10 });
  await page.mouse.up();

  // Wait for pan to complete
  await page.waitForTimeout(300);

  // End performance recording
  const panMetrics = await page.evaluate(() => {
    window.performance.mark('panEnd');
    window.performance.measure('panDuration', 'panStart', 'panEnd');
    const measure = window.performance.getEntriesByName('panDuration')[0];
    return {
      duration: measure.duration,
      memory: window.performance.memory ?
        window.performance.memory.usedJSHeapSize / (1024 * 1024) : 'N/A'
```

```
    };
  });

  console.log('Pan duration (ms):', panMetrics.duration);
  console.log('Memory after pan (MB):', panMetrics.memory);

  // Test with large dataset
  console.log('Testing with large dataset...');

  // Trigger loading of large dataset
  await page.click('#load-large-dataset-btn');
  await page.waitForSelector('#dataset-loaded-indicator');

  // Measure performance after loading large dataset
  const largeDatasetPerformance = await page.evaluate(() => {
    return {
      fps: window.mapPerformanceStats ? window.mapPerformanceStats.fps : 'N/A',
      memory: window.performance.memory ?
        window.performance.memory.usedJSHeapSize / (1024 * 1024) : 'N/A'
    };
  });

  console.log('FPS with large dataset:', largeDatasetPerformance.fps);
  console.log('Memory with large dataset (MB):', largeDatasetPerformance.memory);

  await browser.close();
})();`}
          </pre>
        </div>

        <div>
          <h5>Performance Testing Focus Areas</h5>
          <ul>
            <li>
              <strong>Initial Load Time</strong>: How long until the map is
              interactive
            </li>
            <li>
              <strong>Interaction Responsiveness</strong>: FPS during pan/zoom
            </li>
            <li>
              <strong>Memory Usage</strong>: Especially with large datasets
            </li>
            <li>
              <strong>Network Efficiency</strong>: Tile loading, API calls
            </li>
            <li>
              <strong>CPU/GPU Usage</strong>: During complex operations
            </li>
            <li>
              <strong>Load Testing</strong>: With progressively larger datasets
            </li>
            <li>
              <strong>Mobile Performance</strong>: On lower-powered devices
```

```
            </li>
          </ul>
        </div>
      </div>

      <div>
        <h4>Mocking External Dependencies</h4>

        <p>
          Maps often rely on external services (tile servers, geocoding APIs,
          etc.) that need to be mocked for reliable testing.
        </p>

        <div style={{ marginBottom: '15px' }}>
          <h5>Mocking Tile Layers</h5>
          <pre
            style={{
              backgroundColor: '#f5f5f5',
              padding: '15px',
              borderRadius: '4px',
              overflowX: 'auto',
            }}
          >
            {`// Mock tile layer for testing
const mockTileLayer = {
  getTileUrl: (coords) => {
    // Return a consistent test image instead of actual tiles
    return 'test-assets/mock-tile.png';
  },
  createTile: (coords, done) => {
    const tile = document.createElement('img');
    tile.src = 'test-assets/mock-tile.png';
    tile.width = 256;
    tile.height = 256;
    setTimeout(() => {
      done(null, tile);
    }, 10);
    return tile;
  }
};

// Use it in tests
jest.mock('react-leaflet', () => {
  const actual = jest.requireActual('react-leaflet');
  return {
    ...actual,
    TileLayer: jest.fn(props => {
      // Return mock implementation
      return <div data-testid="mock-tile-layer" />;
    })
  };
});`}
          </pre>
        </div>
```

```
        <div style={{ marginBottom: '15px' }}>
          <h5>Mocking Geolocation</h5>
          <pre
            style={{
              backgroundColor: '#f5f5f5',
              padding: '15px',
              borderRadius: '4px',
              overflowX: 'auto',
            }}
          >
            {`// Mock geolocation in tests
const mockGeolocation = {
  getCurrentPosition: jest.fn().mockImplementation((success) => {
    success({
      coords: {
        latitude: 51.505,
        longitude: -0.09,
        accuracy: 10
      }
    });
  }),
  watchPosition: jest.fn().mockImplementation((success) => {
    // Simulate position updates
    success({
      coords: {
        latitude: 51.505,
        longitude: -0.09,
        accuracy: 10
      }
    });

    // Return mock watch id
    return 123;
  }),
  clearWatch: jest.fn()
};

// Apply mock before tests
global.navigator.geolocation = mockGeolocation;`}
          </pre>
        </div>

        <div>
          <h5>Mocking External APIs</h5>
          <pre
            style={{
              backgroundColor: '#f5f5f5',
              padding: '15px',
              borderRadius: '4px',
              overflowX: 'auto',
            }}
          >
            {`// Mock fetch for API requests
```

```
jest.mock('node-fetch', () => jest.fn());
const fetch = require('node-fetch');

// Set up mock response for geocoding API
fetch.mockImplementation((url) => {
  if (url.includes('geocode')) {
    return Promise.resolve({
      ok: true,
      json: () => Promise.resolve({
        features: [
          {
            geometry: {
              coordinates: [-0.09, 51.505]
            },
            properties: {
              name: 'London',
              country: 'United Kingdom'
            }
          }
        ]
      })
    });
  }

  // Mock responses for other APIs...

  return Promise.resolve({
    ok: false,
    status: 404,
    statusText: 'Not Found'
  });
});`}
          </pre>
        </div>
      </div>
    </div>
  );
};

export default TestingStrategiesGuide;
```

Reference

Common Methods and Properties

Here's a reference guide for frequently used Leaflet methods and properties when working with React:

```
import React from 'react';

const LeafletReferenceGuide = () => {
  return (
```

```
    <div>
      <h3>Common Leaflet Methods and Properties Reference</h3>

      <div style={{ marginBottom: '25px' }}>
        <h4>Map Methods</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Method</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Description</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Example</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>setView(center, zoom)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Sets the
view of the map with the given center and zoom level</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>map.setView([51.505, -0.09], 13)</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>flyTo(center, zoom)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Animates
the map to the given center and zoom level with a smooth pan-zoom</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>map.flyTo([51.51, -0.1], 14)</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>setZoom(zoom)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Sets the
zoom level of the map</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>map.setZoom(15)</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>fitBounds(bounds)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Sets the
view of the map to contain the given bounds</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>map.fitBounds([[51.49, -0.12], [51.52, -0.05]])</code>
              </td>
            </tr>
            <tr>
```

```
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>panTo(center)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Pans the
map to the given center</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>map.panTo([51.505, -0.09])</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>getCenter()
</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Returns the
current center of the map</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>const center = map.getCenter()</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>getZoom()
</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Returns the
current zoom level of the map</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>const zoom = map.getZoom()</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>getBounds()
</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Returns the
current bounds of the map</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>const bounds = map.getBounds()</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>invalidateSize()</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Checks if
the map container size changed and updates the map if so</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>map.invalidateSize()</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>locate(options)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Tries to
locate the user using the Geolocation API</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>map.locate({setView: true, maxZoom: 16})</code>
              </td>
            </tr>
          </tbody>
```

```
        </table>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Layer Methods</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Method</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Description</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Example</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>addTo(map)
</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Adds the
layer to the given map</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>marker.addTo(map)</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>remove()
</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Removes the
layer from the map it is active on</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>marker.remove()</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>bindPopup(content)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Binds a
popup with the given content to the layer</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>marker.bindPopup("Hello")</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>bindTooltip(content)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Binds a
tooltip with the given content to the layer</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>marker.bindTooltip("Tooltip")</code>
              </td>
            </tr>
            <tr>
```

```
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>openPopup()
</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Opens the
bound popup</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>marker.openPopup()</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>closePopup()</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Closes the
bound popup</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>marker.closePopup()</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>setStyle(style)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Changes the
appearance of a path</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>polygon.setStyle({color: 'red'})</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>bringToFront()</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Brings the
layer to the top of all overlays</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>polygon.bringToFront()</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>bringToBack()</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Brings the
layer to the bottom of all overlays</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>polygon.bringToBack()</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>setLatLng(latlng)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Sets the
position of a marker</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>marker.setLatLng([51.5, -0.09])</code>
              </td>
            </tr>
          </tbody>
```

```
        </table>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Event Handling</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Method</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Description</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Example</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>on(eventName, handler)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Adds an
event listener to the object</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>map.on('click', e => console.log(e.latlng))</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>off(eventName, handler)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Removes an
event listener from the object</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>map.off('click', myHandler)</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>once(eventName, handler)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Adds an
event listener that's called only once</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>map.once('zoom', () => console.log('zoomed'))</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>fire(eventName, data)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Fires an
event on the object</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>map.fire('click', {latlng: L.latLng(51.5, -0.09)})</code>
              </td>
            </tr>
          </tbody>
```

```
        </table>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Common Events</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Event</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Description</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Object</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>click</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When the
user clicks on the object</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map,
Layer</td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>dblclick</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When the
user double-clicks on the object</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map,
Layer</td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>mousedown</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When the
user pushes the mouse button on the object</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map,
Layer</td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>mouseover</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When the
mouse enters the object</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map,
Layer</td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>mouseout</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When the
mouse leaves the object</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map,
```

```
Layer</td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>movestart</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When the
user starts moving the map</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map</td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>moveend</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When the
user stops moving the map</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map</td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>zoomstart</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When a zoom
animation starts</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map</td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>zoomend</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When a zoom
animation ends</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map</td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>popupopen</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When a
popup is opened</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map,
Layer</td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>popupclose</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When a
popup is closed</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map,
Layer</td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>locationfound</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When
geolocation is successful</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map</td>
            </tr>
            <tr>
```

```
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>locationerror</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>When
geolocation fails</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Map</td>
            </tr>
          </tbody>
        </table>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>React-Leaflet Hooks</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Hook</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Description</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Example</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>useMap()
</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Access the
Leaflet map instance in any component within the MapContainer</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>const map = useMap()</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>useMapEvent(eventName, handler)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Register an
event on the map instance and listen to it</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>useMapEvent('click', (e) => console.log(e.latlng))</code>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd'
}}>useMapEvents(handlers)</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>Register
multiple events on the map instance</td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <code>{`useMapEvents({
  click: (e) => console.log(e.latlng),
  zoom: () => console.log('zoom')
})`}</code>
              </td>
            </tr>
```

```
          </tbody>
        </table>
      </div>
    </div>
  );
};

export default LeafletReferenceGuide;
```

Useful Plugins

Extend Leaflet's functionality with these popular plugins:

```
import React from 'react';

const LeafletPluginsGuide = () => {
  return (
    <div>
      <h3>Useful Leaflet Plugins</h3>
      <p>
        Enhance your maps with these powerful plugins that extend Leaflet's core
functionality.
      </p>

      <div style={{ marginBottom: '25px' }}>
        <h4>Essential Plugins for React-Leaflet</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd', width: '25%' }}>Plugin</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd', width: '25%' }}>Purpose</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd', width: '50%' }}>Integration Example</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>react-leaflet-markercluster</strong><br />
                <a href="https://github.com/yuzhva/react-leaflet-markercluster"
target="_blank" rel="noopener noreferrer">GitHub</a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Group markers that are close together for better performance and
readability
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`import MarkerClusterGroup from 'react-leaflet-markercluster';
```

```
import 'react-leaflet-markercluster/dist/styles.min.css';

// In your component
<MapContainer>
  <TileLayer ... />
  <MarkerClusterGroup>
    {markers.map(marker => (
      <Marker key={marker.id} position={marker.position}>
        <Popup>{marker.name}</Popup>
      </Marker>
    ))}
  </MarkerClusterGroup>
</MapContainer>`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>react-leaflet-draw</strong><br />
                <a href="https://github.com/alex3165/react-leaflet-draw"
target="_blank" rel="noopener noreferrer">GitHub</a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Allow users to draw and edit shapes on the map
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`import { EditControl } from 'react-leaflet-draw';
import 'leaflet-draw/dist/leaflet.draw.css';

// In your component
<MapContainer>
  <TileLayer ... />
  <FeatureGroup>
    <EditControl
      position="topright"
      onCreated={(e) => console.log(e.layer.toGeoJSON())}
      draw={{
        rectangle: true,
        polygon: true,
        circle: true,
        marker: true
      }}
    />
  </FeatureGroup>
</MapContainer>`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>react-leaflet-heatmap-layer</strong><br />
                <a href="https://github.com/OpenGov/react-leaflet-heatmap-layer"
target="_blank" rel="noopener noreferrer">GitHub</a>
```

```
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  Create heatmaps to visualize density of point data
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`import HeatmapLayer from 'react-leaflet-heatmap-layer';

// In your component
<MapContainer>
  <TileLayer ... />
  <HeatmapLayer
    points={points}
    longitudeExtractor={m => m[1]}
    latitudeExtractor={m => m[0]}
    intensityExtractor={m => m[2]}
    radius={20}
    blur={15}
    max={1.0}
  />
</MapContainer>`}
                  </pre>
                </td>
              </tr>
              <tr>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <strong>react-leaflet-fullscreen</strong><br />
                  <a href="https://github.com/alex3165/react-leaflet-fullscreen"
target="_blank" rel="noopener noreferrer">GitHub</a>
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  Add fullscreen control to expand the map to fill the entire screen
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`import FullscreenControl from 'react-leaflet-fullscreen';
import 'react-leaflet-fullscreen/dist/styles.css';

// In your component
<MapContainer>
  <TileLayer ... />
  <FullscreenControl position="topright" />
</MapContainer>`}
                  </pre>
                </td>
              </tr>
              <tr>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <strong>react-leaflet-search</strong><br />
                  <a href="https://github.com/tumerorkun/react-leaflet-search"
target="_blank" rel="noopener noreferrer">GitHub</a>
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  Add a search control to find locations on the map
```

```
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`import { ReactLeafletSearch } from 'react-leaflet-search';

// In your component
<MapContainer>
  <TileLayer ... />
  <ReactLeafletSearch
    position="topleft"
    provider="OpenStreetMap"
    providerOptions={{ region: 'gb' }}
    inputPlaceholder="Search for a location"
    showMarker={true}
    showPopup={true}
    popUp={(info) => <div>{info.latLng.toString()}</div>}
    closeResultsOnClick={true}
  />
</MapContainer>`}
                </pre>
              </td>
            </tr>
          </tbody>
        </table>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Advanced Visualization Plugins</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd', width: '25%' }}>Plugin</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd', width: '25%' }}>Purpose</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd', width: '50%' }}>Integration Example</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Leaflet.heat</strong><br />
                <a href="https://github.com/Leaflet/Leaflet.heat" target="_blank"
rel="noopener noreferrer">GitHub</a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Create high-performance heatmaps from point data
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`import L from 'leaflet';
import 'leaflet.heat';
import { useMap } from 'react-leaflet';
```

```
// Create a component to add the heatmap
function HeatmapLayer({ points }) {
  const map = useMap();

  useEffect(() => {
    if (!map) return;

    const heatLayer = L.heatLayer(points, {
      radius: 25,
      blur: 15,
      maxZoom: 17
    }).addTo(map);

    return () => {
      map.removeLayer(heatLayer);
    };
  }, [map, points]);

  return null;
}`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Leaflet.VectorGrid</strong><br />
                <a href="https://github.com/Leaflet/Leaflet.VectorGrid"
target="_blank" rel="noopener noreferrer">GitHub</a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Display vector tiles for high-performance visualization of large
GeoJSON datasets
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`import L from 'leaflet';
import 'leaflet.vectorgrid';
import { useMap } from 'react-leaflet';

// Create a component to add vector tiles
function VectorTileLayer({ url, options }) {
  const map = useMap();

  useEffect(() => {
    if (!map) return;

    // Create the vector tile layer
    const vectorTileLayer = L.vectorGrid.protobuf(url, {
      vectorTileLayerStyles: {
        water: {
          fill: true,
          fillColor: '#2375c3',
          fillOpacity: 0.5,
```

```
          weight: 1,
          color: '#000000'
        },
        // Styles for other features...
        ...options.styles
      },
      ...options
    }).addTo(map);

    return () => {
      map.removeLayer(vectorTileLayer);
    };
  }, [map, url, options]);

  return null;
}`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Leaflet.Editable</strong><br />
                <a href="https://github.com/Leaflet/Leaflet.Editable"
target="_blank" rel="noopener noreferrer">GitHub</a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Provide advanced editing capabilities for vector features
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`import L from 'leaflet';
import 'leaflet-editable';
import { useMap } from 'react-leaflet';

// Create a component to enable editing
function EditableMapFeatures() {
  const map = useMap();

  useEffect(() => {
    if (!map) return;

    // Enable editing
    map.editable = true;

    // Create an editable polygon
    const polygon = map.editTools.startPolygon();

    // Listen for changes
    polygon.on('editable:editing', (e) => {
      console.log('Polygon updated:', polygon.getLatLngs());
    });

    return () => {
      map.editTools.stopDrawing();
```

```
      map.removeLayer(polygon);
    };
  }, [map]);

  return null;
}`}
                </pre>
              </td>
            </tr>
          </tbody>
        </table>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Routing and Geocoding Plugins</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd', width: '25%' }}>Plugin</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd', width: '25%' }}>Purpose</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd', width: '50%' }}>Integration Example</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Leaflet Routing Machine</strong><br />
                <a href="https://github.com/perliedman/leaflet-routing-machine"
target="_blank" rel="noopener noreferrer">GitHub</a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Calculate and display routes between locations
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`import L from 'leaflet';
import 'leaflet-routing-machine';
import 'leaflet-routing-machine/dist/leaflet-routing-machine.css';
import { useMap } from 'react-leaflet';

// Create a component to add routing
function RoutingMachine({ waypoints }) {
  const map = useMap();

  useEffect(() => {
    if (!map || !waypoints.length) return;

    // Create the routing control
    const routingControl = L.Routing.control({
      waypoints: waypoints.map(wp => L.latLng(wp[0], wp[1])),
      routeWhileDragging: true,
```

```
      showAlternatives: true,
      lineOptions: {
        styles: [{ color: '#6FA1EC', weight: 4 }]
      },
      altLineOptions: {
        styles: [{ color: '#cdcdcd', weight: 4 }]
      }
    }).addTo(map);

    return () => {
      map.removeControl(routingControl);
    };
  }, [map, waypoints]);

  return null;
}`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Leaflet Control Geocoder</strong><br />
                <a href="https://github.com/perliedman/leaflet-control-geocoder"
target="_blank" rel="noopener noreferrer">GitHub</a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Add a geocoding control for address search
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`import L from 'leaflet';
import 'leaflet-control-geocoder';
import 'leaflet-control-geocoder/dist/Control.Geocoder.css';
import { useMap } from 'react-leaflet';

// Create a component to add geocoding
function GeocoderControl() {
  const map = useMap();

  useEffect(() => {
    if (!map) return;

    // Create the geocoder control
    const geocoder = L.Control.geocoder({
      defaultMarkGeocode: false,
      position: 'topleft',
      placeholder: 'Search for a location...',
      errorMessage: 'Nothing found.'
    }).on('markgeocode', function(e) {
      const { center, bbox } = e.geocode;

      // Create a marker at the found location
      L.marker(center).addTo(map)
        .bindPopup(e.geocode.name)
```

```
          .openPopup();

        // Zoom to the bounding box
        map.fitBounds(bbox);
      }).addTo(map);

      return () => {
        map.removeControl(geocoder);
      };
    }, [map]);

    return null;
}`}
                  </pre>
                </td>
              </tr>
              <tr>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <strong>Leaflet.Locate</strong><br />
                  <a href="https://github.com/domoritz/leaflet-locatecontrol"
target="_blank" rel="noopener noreferrer">GitHub</a>
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  Add a control to locate and track the user's position
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`import L from 'leaflet';
import 'leaflet.locatecontrol';
import 'leaflet.locatecontrol/dist/L.Control.Locate.min.css';
import { useMap } from 'react-leaflet';

// Create a component to add location control
function LocateControl() {
  const map = useMap();

  useEffect(() => {
    if (!map) return;

    // Create the locate control
    const locateControl = L.control.locate({
      position: 'topright',
      setView: 'untilPanOrZoom',
      flyTo: true,
      cacheLocation: true,
      drawCircle: true,
      drawMarker: true,
      showPopup: true,
      strings: {
        title: 'Show my location',
        popup: 'You are within {distance} {unit} from this point',
        outsideMapBoundsMsg: 'You seem to be outside the map boundaries'
      },
      locateOptions: {
```

```
        enableHighAccuracy: true,
        maxZoom: 16
      }
    }).addTo(map);

    return () => {
      map.removeControl(locateControl);
    };
  }, [map]);

  return null;
}`}
                </pre>
              </td>
            </tr>
          </tbody>
        </table>
      </div>

      <div>
        <h4>Using Plugins with React-Leaflet</h4>
        <p>
          When integrating Leaflet plugins with React-Leaflet, follow these
guidelines:
        </p>
        <ol>
          <li>
            <strong>Check for React wrappers first</strong> - Many popular Leaflet
plugins have dedicated React-Leaflet wrappers that make integration easier.
          </li>
          <li>
            <strong>Use the useMap hook</strong> - For plugins without React
wrappers, create a custom component using the useMap hook to access the map
instance.
          </li>
          <li>
            <strong>Properly cleanup in useEffect</strong> - Always remove
controls, layers, or event listeners in the cleanup function of useEffect to
prevent memory leaks.
          </li>
          <li>
            <strong>Import CSS</strong> - Don't forget to import any necessary CSS
files for plugins that include UI components.
          </li>
          <li>
            <strong>Handle state properly</strong> - When plugin state needs to be
synchronized with React state, use refs and effect hooks to maintain consistency.
          </li>
        </ol>
        <p>
          <strong>Example pattern for a custom plugin wrapper:</strong>
        </p>
        <pre style={{
          backgroundColor: '#f5f5f5',
```

```
            padding: '15px',
            borderRadius: '4px',
            overflowX: 'auto',
            marginTop: '10px'
          }}>
{`import React, { useEffect } from 'react';
import { useMap } from 'react-leaflet';
import L from 'leaflet';
import 'path/to/plugin';
import 'path/to/plugin.css';

const CustomPluginComponent = ({ options, ...props }) => {
  const map = useMap();

  useEffect(() => {
    // Initialize the plugin
    const pluginInstance = L.somePlugin(options).addTo(map);

    // Set up event listeners
    pluginInstance.on('someevent', props.onSomeEvent);

    // Return cleanup function
    return () => {
      pluginInstance.off('someevent', props.onSomeEvent);
      map.removeLayer(pluginInstance);  // or removeControl for controls
    };
  }, [map, options, props.onSomeEvent]);

  return null;  // This component doesn't render anything itself
};

export default CustomPluginComponent;

// Usage in parent component:
// <MapContainer>
//   <TileLayer ... />
//   <CustomPluginComponent
//     options={{ ... }}
//     onSomeEvent={handleEvent}
//   />
// </MapContainer>`}
        </pre>
      </div>
    </div>
  );
```

Map Tiles and GeoJSON Resources

A collection of resources for finding map tiles and geographical data:

```
import React from 'react';

const MapResourcesGuide = () => {
  return (
    <div>
      <h3>Map Tiles and GeoJSON Resources</h3>
      <p>
        Finding the right map tiles and geographical data is essential for
        creating effective maps. Here's a comprehensive collection of resources
        to help you build feature-rich maps.
      </p>

      <div style={{ marginBottom: '25px' }}>
        <h4>Free Tile Providers</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
                  width: '25%',
                }}
              >
                Provider
              </th>
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
                  width: '25%',
                }}
              >
                Features
              </th>
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
                  width: '50%',
                }}
              >
                Usage Example
              </th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>OpenStreetMap</strong>
```

```
                  <br />
                  <a
                    href="https://www.openstreetmap.org/"
                    target="_blank"
                    rel="noopener noreferrer"
                  >
                    Website
                  </a>
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <ul style={{ margin: 0, paddingLeft: '20px' }}>
                    <li>Community-maintained world map</li>
                    <li>Standard map style</li>
                    <li>Free for non-commercial use</li>
                    <li>Usage limits apply</li>
                  </ul>
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
                    {`<TileLayer
  url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
  attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
/>`}
                  </pre>
                </td>
              </tr>
              <tr>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <strong>CartoDB</strong>
                  <br />
                  <a
                    href="https://carto.com/basemaps/"
                    target="_blank"
                    rel="noopener noreferrer"
                  >
                    Website
                  </a>
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <ul style={{ margin: 0, paddingLeft: '20px' }}>
                    <li>Beautiful basemaps</li>
                    <li>Multiple styles</li>
                    <li>Light, dark, and voyager themes</li>
                    <li>Free tier with limitations</li>
                  </ul>
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
                    {`// Light theme
<TileLayer
  url="https://{s}.basemaps.cartocdn.com/light_all/{z}/{x}/{y}{r}.png"
  attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors
```

```
&copy; <a href="https://carto.com/attributions">CARTO</a>'
  subdomains="abcd"
  maxZoom={19}
/>

// Dark theme
<TileLayer
  url="https://{s}.basemaps.cartocdn.com/dark_all/{z}/{x}/{y}{r}.png"
  attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors
&copy; <a href="https://carto.com/attributions">CARTO</a>'
  subdomains="abcd"
  maxZoom={19}
/>`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Stamen</strong>
                <br />
                <a
                  href="http://maps.stamen.com/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  Website
                </a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Artistic map styles</li>
                  <li>Terrain, Toner, and Watercolor</li>
                  <li>Great for unique visualizations</li>
                  <li>Free for non-commercial use</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
                  {`// Terrain style
<TileLayer
  url="https://stamen-tiles-{s}.a.ssl.fastly.net/terrain/{z}/{x}/{y}{r}.png"
  attribution='Map tiles by <a href="http://stamen.com">Stamen Design</a>, <a
href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a> &mdash; Map data
&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
  subdomains="abcd"
  minZoom={0}
  maxZoom={18}
/>

// Watercolor style
<TileLayer
  url="https://stamen-tiles-{s}.a.ssl.fastly.net/watercolor/{z}/{x}/{y}.jpg"
```

```
  attribution='Map tiles by <a href="http://stamen.com">Stamen Design</a>, <a
href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a> &mdash; Map data
&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
  subdomains="abcd"
  minZoom={1}
  maxZoom={16}
/>`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Thunderforest</strong>
                <br />
                <a
                  href="https://www.thunderforest.com/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  Website
                </a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Specialized maps</li>
                  <li>Outdoors, transport, landscape</li>
                  <li>Free tier with API key</li>
                  <li>Requires registration</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
                  {`// Outdoors style (requires API key)
<TileLayer
  url="https://{s}.tile.thunderforest.com/outdoors/{z}/{x}/{y}.png?
apikey=YOUR_API_KEY"
  attribution='&copy; <a href="http://www.thunderforest.com/">Thunderforest</a>,
&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
  subdomains="abcd"
  maxZoom={22}
/>`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>ESRI</strong>
                <br />
                <a
                  href="https://www.arcgis.com/home/webmap/viewer.html"
                  target="_blank"
                  rel="noopener noreferrer"
```

```
                >
                  Website
                </a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>High-quality basemaps</li>
                  <li>Satellite imagery</li>
                  <li>World imagery and topographic maps</li>
                  <li>Free tier with attribution</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
                  {`// World Imagery (satellite)
<TileLayer

url="https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/
tile/{z}/{y}/{x}"
  attribution='Tiles &copy; Esri &mdash; Source: Esri, i-cubed, USDA, USGS, AEX,
GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community'
/>

// Topographic
<TileLayer

url="https://server.arcgisonline.com/ArcGIS/rest/services/World_Topo_Map/MapServer
/tile/{z}/{y}/{x}"
  attribution='Tiles &copy; Esri &mdash; Esri, DeLorme, NAVTEQ, TomTom, Intermap,
iPC, USGS, FAO, NPS, NRCAN, GeoBase, Kadaster NL, Ordnance Survey, Esri Japan,
METI, Esri China (Hong Kong), and the GIS User Community'
/>`}
                </pre>
              </td>
            </tr>
          </tbody>
        </table>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Commercial Tile Providers</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
                  width: '25%',
                }}
              >
                Provider
              </th>
```

```
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
                  width: '25%',
                }}
              >
                Features
              </th>
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
                  width: '50%',
                }}
              >
                Usage Notes
              </th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Mapbox</strong>
                <br />
                <a
                  href="https://www.mapbox.com/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  Website
                </a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Highly customizable styles</li>
                  <li>Vector tiles for smooth zooming</li>
                  <li>Studio for custom design</li>
                  <li>Rich APIs for routing, geocoding</li>
                  <li>Free tier available</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <p>Requires an access token.</p>
                <pre style={{ margin: '5px 0 0 0', whiteSpace: 'pre-wrap' }}>
                  {`<TileLayer
  url="https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token=
{accessToken}"
  attribution='Map data &copy; <a
href="https://www.openstreetmap.org/">OpenStreetMap</a> contributors, <a
href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, Imagery © <a
href="https://www.mapbox.com/">Mapbox</a>'
```

```
    id="mapbox/streets-v11" // or satellite-v9, light-v10, dark-v10, outdoors-v11
    accessToken="YOUR_ACCESS_TOKEN"
    tileSize={512}
    zoomOffset={-1}
    maxZoom={19}
  />`}
                </pre>
                <p>
                  For vector tiles, consider using Mapbox GL JS directly or
                  react-map-gl.
                </p>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Google Maps</strong>
                <br />
                <a
                  href="https://cloud.google.com/maps-platform/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  Website
                </a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Familiar map style</li>
                  <li>Excellent global coverage</li>
                  <li>Street View integration</li>
                  <li>API key required</li>
                  <li>Pay-as-you-go pricing</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <p>
                  For Google Maps with React, use the official
                  @react-google-maps/api package or google-map-react.
                </p>
                <p>
                  Google Maps cannot be used directly with Leaflet due to terms
                  of service. If you want to use Google Maps, use their official
                  SDK instead of Leaflet.
                </p>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Maptiler</strong>
                <br />
                <a
                  href="https://www.maptiler.com/"
                  target="_blank"
                  rel="noopener noreferrer"
```

```
                >
                  Website
                </a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Vector and raster tiles</li>
                  <li>Various styles available</li>
                  <li>Custom styling tools</li>
                  <li>Free tier with watermark</li>
                  <li>Affordable paid plans</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
                  {`<TileLayer
  url="https://api.maptiler.com/maps/streets/{z}/{x}/{y}.png?key=YOUR_API_KEY"
  attribution='<a href="https://www.maptiler.com/copyright/"
target="_blank">&copy; MapTiler</a> <a
href="https://www.openstreetmap.org/copyright" target="_blank">&copy;
OpenStreetMap contributors</a>'
  tileSize={512}
  zoomOffset={-1}
  minZoom={1}
  maxZoom={19}
/>`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>TomTom</strong>
                <br />
                <a
                  href="https://developer.tomtom.com/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  Website
                </a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>High-quality road maps</li>
                  <li>Excellent for routing applications</li>
                  <li>Traffic data integration</li>
                  <li>Free tier available</li>
                  <li>API key required</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
                  {`<TileLayer
  url="https://{s}.api.tomtom.com/map/1/tile/basic/{style}/{z}/{x}/{y}.png?
```

```
key=YOUR_API_KEY"
  attribution='<a href="https://tomtom.com" target="_blank">&copy; TomTom</a>'
  subdomains="abcd"
  style="main" // or night
  minZoom={1}
  maxZoom={22}
/>`}
                </pre>
              </td>
            </tr>
          </tbody>
        </table>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>GeoJSON Data Sources</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
                }}
              >
                Source
              </th>
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
                }}
              >
                Description
              </th>
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
                }}
              >
                URL
              </th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Natural Earth</strong>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
```

```
            Public domain map dataset with country boundaries, populated
            places, roads, etc. Available in multiple scales.
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <a
              href="https://www.naturalearthdata.com/"
              target="_blank"
              rel="noopener noreferrer"
            >
              https://www.naturalearthdata.com/
            </a>
          </td>
        </tr>
        <tr>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <strong>GeoJSON.io</strong>
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            Web-based tool for creating, viewing, and sharing GeoJSON. Great
            for creating small custom datasets.
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <a
              href="https://geojson.io/"
              target="_blank"
              rel="noopener noreferrer"
            >
              https://geojson.io/
            </a>
          </td>
        </tr>
        <tr>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <strong>OpenStreetMap Extracts</strong>
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            Download OpenStreetMap data for specific regions in various
            formats, including GeoJSON.
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <a
              href="https://download.geofabrik.de/"
              target="_blank"
              rel="noopener noreferrer"
            >
              https://download.geofabrik.de/
            </a>
          </td>
        </tr>
        <tr>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <strong>USGS</strong>
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
```

```
            U.S. Geological Survey data portal with various geographical
            datasets including boundaries, elevation, hydrography.
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <a
              href="https://www.usgs.gov/products/data"
              target="_blank"
              rel="noopener noreferrer"
            >
              https://www.usgs.gov/products/data
            </a>
          </td>
        </tr>
        <tr>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <strong>NOAA</strong>
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            National Oceanic and Atmospheric Administration data including
            weather, climate, oceans, and coastlines.
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <a
              href="https://www.ncei.noaa.gov/maps/geoportal/"
              target="_blank"
              rel="noopener noreferrer"
            >
              https://www.ncei.noaa.gov/maps/geoportal/
            </a>
          </td>
        </tr>
        <tr>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <strong>Overpass Turbo</strong>
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            Web-based data filtering tool for OpenStreetMap. Create queries
            and export results as GeoJSON.
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <a
              href="https://overpass-turbo.eu/"
              target="_blank"
              rel="noopener noreferrer"
            >
              https://overpass-turbo.eu/
            </a>
          </td>
        </tr>
        <tr>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <strong>Data.gov</strong>
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
```

```
                U.S. government's open data portal with thousands of datasets,
                many with geographical components.
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <a
                  href="https://www.data.gov/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  https://www.data.gov/
                </a>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>European Data Portal</strong>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Open data from European public administrations, including
                geographical data.
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <a
                  href="https://data.europa.eu/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  https://data.europa.eu/
                </a>
              </td>
            </tr>
          </tbody>
        </table>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Tools for GeoJSON Processing</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
                }}
              >
                Tool
              </th>
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
```

```
            }}
          >
            Purpose
          </th>
          <th
            style={{
              padding: '8px',
              textAlign: 'left',
              border: '1px solid #ddd',
            }}
          >
            URL
          </th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <strong>Turf.js</strong>
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            JavaScript library for spatial analysis. Useful for modifying,
            combining, and analyzing GeoJSON data.
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <a
              href="https://turfjs.org/"
              target="_blank"
              rel="noopener noreferrer"
            >
              https://turfjs.org/
            </a>
          </td>
        </tr>
        <tr>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <strong>QGIS</strong>
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            Free, open-source geographic information system for viewing,
            editing, and analyzing geospatial data.
          </td>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
            <a
              href="https://qgis.org/"
              target="_blank"
              rel="noopener noreferrer"
            >
              https://qgis.org/
            </a>
          </td>
        </tr>
        <tr>
          <td style={{ padding: '8px', border: '1px solid #ddd' }}>
```

```
                <strong>MapShaper</strong>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Web-based tool for simplifying and converting GeoJSON and other
                formats. Great for reducing file size.
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <a
                  href="https://mapshaper.org/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  https://mapshaper.org/
                </a>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>GDAL/OGR</strong>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Command-line tools for manipulating geospatial data formats.
                Powerful for batch processing.
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <a
                  href="https://gdal.org/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  https://gdal.org/
                </a>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Tippecanoe</strong>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Tool for converting large GeoJSON files into vector tiles.
                Useful for optimizing performance with large datasets.
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <a
                  href="https://github.com/mapbox/tippecanoe"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  GitHub: mapbox/tippecanoe
                </a>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
```

```
                <strong>geojson.io</strong>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Web-based editor for creating and editing GeoJSON data visually.
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <a
                  href="https://geojson.io/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  https://geojson.io/
                </a>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>GeoJSON Lint</strong>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                Validates GeoJSON data to ensure it follows the specification
                correctly.
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <a
                  href="https://geojsonlint.com/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  https://geojsonlint.com/
                </a>
              </td>
            </tr>
          </tbody>
        </table>
      </div>

      <div>
        <h4>Useful APIs for Mapping Projects</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
                }}
              >
                API
              </th>
              <th
                style={{
                  padding: '8px',
```

```
                  textAlign: 'left',
                  border: '1px solid #ddd',
                }}
              >
                Features
              </th>
              <th
                style={{
                  padding: '8px',
                  textAlign: 'left',
                  border: '1px solid #ddd',
                }}
              >
                Notes
              </th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Nominatim</strong>
                <br />
                <a
                  href="https://nominatim.org/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  Website
                </a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Geocoding (address to coordinates)</li>
                  <li>Reverse geocoding (coordinates to address)</li>
                  <li>Based on OpenStreetMap data</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <p>
                  Free API with usage limits. For production use, consider
                  self-hosting or using a commercial service.
                </p>
                <p>Example usage (forward geocoding):</p>
                <pre style={{ margin: '5px 0 0 0', whiteSpace: 'pre-wrap' }}>
                  {`fetch('https://nominatim.openstreetmap.org/search?
format=json&q=London')
  .then(response => response.json())
  .then(data => console.log(data));`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>OpenRouteService</strong>
```

```
                <br />
                <a
                  href="https://openrouteservice.org/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  Website
                </a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Routing (directions)</li>
                  <li>Isochrones (travel time areas)</li>
                  <li>Matrix calculations (distances between points)</li>
                  <li>Elevation profiles</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <p>
                  Free tier with API key. Supports various travel modes (car,
                  bike, foot, etc.).
                </p>
                <p>Example usage (routing):</p>
                <pre style={{ margin: '5px 0 0 0', whiteSpace: 'pre-wrap' }}>
                  {`fetch('https://api.openrouteservice.org/v2/directions/driving-
car?api_key=YOUR_API_KEY&start=-0.1,51.5&end=-0.2,51.5')
  .then(response => response.json())
  .then(data => console.log(data));`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Geoapify</strong>
                <br />
                <a
                  href="https://www.geoapify.com/"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  Website
                </a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Geocoding</li>
                  <li>Places API (POI search)</li>
                  <li>Routing</li>
                  <li>Isochrones</li>
                  <li>Static maps</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <p>
```

```
                  Free tier with API key. Comprehensive set of location
                  services.
                </p>
                <p>Example usage (places search):</p>
                <pre style={{ margin: '5px 0 0 0', whiteSpace: 'pre-wrap' }}>
                  {`fetch('https://api.geoapify.com/v2/places?
categories=commercial.supermarket&filter=circle:-0.09,51.5,5000&limit=20&apiKey=YO
UR_API_KEY')
  .then(response => response.json())
  .then(data => console.log(data));`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>OpenWeatherMap</strong>
                <br />
                <a
                  href="https://openweathermap.org/api"
                  target="_blank"
                  rel="noopener noreferrer"
                >
                  Website
                </a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Current weather data</li>
                  <li>Weather forecasts</li>
                  <li>Historical weather data</li>
                  <li>Weather map layers</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <p>
                  Free tier with API key. Great for adding weather data to maps.
                </p>
                <p>Example usage (current weather):</p>
                <pre style={{ margin: '5px 0 0 0', whiteSpace: 'pre-wrap' }}>
                  {`fetch('https://api.openweathermap.org/data/2.5/weather?
lat=51.5&lon=-0.09&appid=YOUR_API_KEY&units=metric')
  .then(response => response.json())
  .then(data => console.log(data));`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Overpass API</strong>
                <br />
                <a
                  href="https://wiki.openstreetmap.org/wiki/Overpass_API"
                  target="_blank"
                  rel="noopener noreferrer"
```

```
                >
                  Website
                </a>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Query OpenStreetMap data</li>
                  <li>Filter by location, tags, and relationships</li>
                  <li>Get detailed POI data</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <p>
                  Free API for querying OSM data. Complex but powerful query
                  language.
                </p>
                <p>Example usage (find restaurants in an area):</p>
                <pre style={{ margin: '5px 0 0 0', whiteSpace: 'pre-wrap' }}>
                  {`const query = \`
[out:json];
node
  ["amenity"="restaurant"]
  (51.49,-0.11,51.52,-0.07);
out;
\`;

fetch(\`https://overpass-api.de/api/interpreter?
data=\${encodeURIComponent(query)}\`)
  .then(response => response.json())
  .then(data => console.log(data));`}
                </pre>
              </td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
  );
};

export default MapResourcesGuide;
```

Debugging Techniques

When working with Leaflet and React, you may encounter various issues. Here are techniques to help you debug and solve common problems:

```
import React from 'react';

const DebuggingTechniquesGuide = () => {
  return (
```

```
    <div>
      <h3>Debugging Techniques for Leaflet with React</h3>
      <p>
        Maps can be complex to debug due to their visual nature and dependencies
on external resources.
        Here's a comprehensive guide to identify and resolve common issues.
      </p>

      <div style={{ marginBottom: '25px' }}>
        <h4>Common Leaflet Issues and Solutions</h4>
        <table style={{ width: '100%', borderCollapse: 'collapse' }}>
          <thead>
            <tr style={{ backgroundColor: '#f5f5f5' }}>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Issue</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Potential Causes</th>
              <th style={{ padding: '8px', textAlign: 'left', border: '1px solid
#ddd' }}>Solutions</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Map not displaying</strong>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Missing CSS</li>
                  <li>Container has no height</li>
                  <li>Invalid center coordinates</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ol style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Ensure you've imported Leaflet CSS: <code>import
'leaflet/dist/leaflet.css';</code></li>
                  <li>Check container dimensions: <code>style={{ height: '500px',
width: '100%' }}</code></li>
                  <li>Verify center coordinates are valid: <code>[51.505, -0.09]
</code> not <code>[-0.09, 51.505]</code></li>
                </ol>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Missing marker icons</strong>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Icon image paths not resolved</li>
                  <li>Webpack/bundler configuration issue</li>
                </ul>
              </td>
```

```
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`// Add this code to fix marker icon issues
import L from 'leaflet';
import icon from 'leaflet/dist/images/marker-icon.png';
import iconShadow from 'leaflet/dist/images/marker-shadow.png';

// Define the default icon
let DefaultIcon = L.icon({
  iconUrl: icon,
  shadowUrl: iconShadow,
  iconSize: [25, 41],
  iconAnchor: [12, 41],
  popupAnchor: [1, -34]
});

// Set the default icon for all markers
L.Marker.prototype.options.icon = DefaultIcon;`}
                  </pre>
                </td>
              </tr>
              <tr>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <strong>Map not updating with props</strong>
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <ul style={{ margin: 0, paddingLeft: '20px' }}>
                    <li>MapContainer doesn't react to prop changes</li>
                    <li>Missing dependency in useEffect</li>
                  </ul>
                </td>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                  <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`// Create a controller component to update the map view
const MapViewController = ({ center, zoom }) => {
  const map = useMap();

  useEffect(() => {
    map.setView(center, zoom);
  }, [map, center, zoom]); // Include all dependencies

  return null;
};

// In your main component
<MapContainer center={initialCenter} zoom={initialZoom}>
  <TileLayer ... />
  <MapViewController center={center} zoom={zoom} />
</MapContainer>`}
                  </pre>
                </td>
              </tr>
              <tr>
                <td style={{ padding: '8px', border: '1px solid #ddd' }}>
```

```
                <strong>Memory leaks</strong>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Event listeners not removed</li>
                  <li>Layers or controls not cleaned up</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <pre style={{ margin: 0, whiteSpace: 'pre-wrap' }}>
{`useEffect(() => {
  // Add event listeners or map objects
  const clickHandler = () => { /* ... */ };
  map.on('click', clickHandler);

  const customLayer = L.layerGroup().addTo(map);

  // Return cleanup function to remove them
  return () => {
    map.off('click', clickHandler);
    map.removeLayer(customLayer);
  };
}, [map]);`}
                </pre>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Tiles not loading</strong>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>CORS issues</li>
                  <li>Incorrect URL format</li>
                  <li>API key issues</li>
                  <li>Rate limiting</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ol style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Check browser console for CORS errors</li>
                  <li>Verify tile URL format (z/x/y parameters)</li>
                  <li>Check API key is valid and included correctly</li>
                  <li>Try a different tile provider to isolate the issue</li>
                  <li>Add <code>{r: ''}</code> to tileLayer options to prevent
retina detection issues</li>
                </ol>
              </td>
            </tr>
            <tr>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <strong>Z-index/layer order problems</strong>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
```

```
                <ul style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Layers rendered in wrong order</li>
                  <li>Markers hidden behind other layers</li>
                </ul>
              </td>
              <td style={{ padding: '8px', border: '1px solid #ddd' }}>
                <ol style={{ margin: 0, paddingLeft: '20px' }}>
                  <li>Use <code>zIndex</code> option on layers</li>
                  <li>Use <code>bringToFront()</code> and <code>bringToBack()
</code> methods</li>
                  <li>Use <code>pane</code> option to control rendering order</li>
                  <li>Create custom panes with <code>map.createPane(name)</code>
and set <code>zIndex</code></li>
                </ol>
              </td>
            </tr>
          </tbody>
        </table>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Debugging Map Interactions</h4>

        <h5>1. Inspect Map Instance</h5>
        <pre style={{ backgroundColor: '#f5f5f5', padding: '15px', borderRadius:
'4px', overflowX: 'auto' }}>
{`// Add this to your component for development
const DebugMap = () => {
  const map = useMap();

  useEffect(() => {
    // Expose map to window for console debugging
    window.debugMap = map;

    // Log map events
    const logEvent = (e) => {
      console.log(\`Map event: \${e.type}\`, e);
    };

    map.on('click moveend zoomend layeradd layerremove', logEvent);

    return () => {
      delete window.debugMap;
      map.off('click moveend zoomend layeradd layerremove', logEvent);
    };
  }, [map]);

  return null;
};

// Then add to your MapContainer
<MapContainer>
  <TileLayer ... />
  {process.env.NODE_ENV === 'development' && <DebugMap />}
```

```
  </MapContainer>

  // Now in browser console you can:
  // Check current map state
  console.log(window.debugMap.getCenter(), window.debugMap.getZoom());
  // Test methods
  window.debugMap.flyTo([51.5, -0.1], 14);
  // Check what layers exist
  console.log(Object.keys(window.debugMap._layers));`}
          </pre>

          <h5>2. Track Layer Events</h5>
          <pre style={{ backgroundColor: '#f5f5f5', padding: '15px', borderRadius:
'4px', overflowX: 'auto' }}>
{`// Component to debug layer interactions
const DebugLayer = ({ layer, name }) => {
  useEffect(() => {
    if (!layer) return;

    // Log all events on this layer
    const events = [
      'click', 'dblclick', 'mousedown', 'mouseup', 'mouseover',
      'mouseout', 'contextmenu', 'dragstart', 'drag', 'dragend'
    ];

    const logEvent = (e) => {
      console.log(\`\${name} event: \${e.type}\`, e);
    };

    events.forEach(event => {
      layer.on(event, logEvent);
    });

    return () => {
      events.forEach(event => {
        layer.off(event, logEvent);
      });
    };
  }, [layer, name]);

  return null;
};

// Usage with a ref to the layer
const MyComponent = () => {
  const markerRef = useRef(null);

  return (
    <>
      <Marker
        position={[51.505, -0.09]}
        ref={markerRef}
      />
      <DebugLayer layer={markerRef.current} name="MyMarker" />
```

```
      </>
    );
  };`}
        </pre>

        <h5>3. Visual Debugging Helpers</h5>
        <pre style={{ backgroundColor: '#f5f5f5', padding: '15px', borderRadius:
'4px', overflowX: 'auto' }}>
{`// Component to visualize bounds or other debug information
const VisualDebugger = ({ bounds }) => {
  const map = useMap();

  useEffect(() => {
    if (!bounds) return;

    // Create a rectangle showing the bounds
    const rectangle = L.rectangle(bounds, {
      color: '#ff7800',
      weight: 2,
      fillOpacity: 0.1,
      dashArray: '5, 5',
      interactive: false
    }).addTo(map);

    // Add text label
    const center = rectangle.getBounds().getCenter();
    const label = L.marker(center, {
      icon: L.divIcon({
        className: 'debug-label',
        html: \`<div style="color:#ff7800;font-weight:bold;">Bounds</div>\`,
        iconSize: [80, 20]
      }),
      interactive: false
    }).addTo(map);

    return () => {
      map.removeLayer(rectangle);
      map.removeLayer(label);
    };
  }, [map, bounds]);

  return null;
};`}
        </pre>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Performance Debugging</h4>

        <h5>1. Measure Render Time</h5>
        <pre style={{ backgroundColor: '#f5f5f5', padding: '15px', borderRadius:
'4px', overflowX: 'auto' }}>
{`// Component to monitor map rendering performance
const PerformanceMonitor = () => {
```

```
    const map = useMap();
    const [stats, setStats] = useState({
      fps: 0,
      renderTime: 0,
      tileCount: 0,
      layerCount: 0
    });

    useEffect(() => {
      if (!map) return;

      let frameCount = 0;
      let lastTime = performance.now();
      let renderStartTime = 0;

      // Track frame rate
      const measureFps = () => {
        const now = performance.now();
        frameCount++;

        if (now - lastTime >= 1000) {
          setStats(prev => ({
            ...prev,
            fps: Math.round(frameCount * 1000 / (now - lastTime))
          }));
          frameCount = 0;
          lastTime = now;
        }

        requestAnimationFrame(measureFps);
      };

      // Track events that may cause rendering
      const trackRenderStart = () => {
        renderStartTime = performance.now();
      };

      const trackRenderEnd = () => {
        if (renderStartTime === 0) return;

        const renderTime = performance.now() - renderStartTime;
        renderStartTime = 0;

        setStats(prev => ({
          ...prev,
          renderTime: renderTime.toFixed(2),
          tileCount: document.querySelectorAll('.leaflet-tile').length,
          layerCount: Object.keys(map._layers || {}).length
        }));
      };

      // Start measuring
      const animationId = requestAnimationFrame(measureFps);
```

```
    map.on('movestart zoomstart', trackRenderStart);
    map.on('moveend zoomend', trackRenderEnd);

    // Create UI to display stats
    const statsDiv = L.DomUtil.create('div', 'leaflet-control leaflet-bar
performance-stats');
    statsDiv.style.cssText = \`
      background: white;
      padding: 6px 8px;
      font-family: monospace;
      font-size: 12px;
      color: #333;
      border: 1px solid #ccc;
      border-radius: 4px;
      min-width: 150px;
      max-width: 300px;
      position: absolute;
      bottom: 20px;
      right: 10px;
      z-index: 1000;
    \`;

    const updateStatsDisplay = () => {
      statsDiv.innerHTML = \`
        <div><strong>FPS:</strong> \${stats.fps}</div>
        <div><strong>Render Time:</strong> \${stats.renderTime}ms</div>
        <div><strong>Tiles:</strong> \${stats.tileCount}</div>
        <div><strong>Layers:</strong> \${stats.layerCount}</div>
      \`;
    };

    map.getContainer().appendChild(statsDiv);

    const displayInterval = setInterval(updateStatsDisplay, 500);

    return () => {
      cancelAnimationFrame(animationId);
      clearInterval(displayInterval);
      map.off('movestart zoomstart', trackRenderStart);
      map.off('moveend zoomend', trackRenderEnd);
      map.getContainer().removeChild(statsDiv);
    };
  }, [map, stats]);

  return null;
};`}
        </pre>

        <h5>2. Memory Usage Monitoring</h5>
        <pre style={{ backgroundColor: '#f5f5f5', padding: '15px', borderRadius:
'4px', overflowX: 'auto' }}>
{`// In Chrome DevTools:
// 1. Open the Performance tab
// 2. Check "Memory" checkbox
```

```
// 3. Click "Record"
// 4. Interact with your map
// 5. Click "Stop"
// 6. Analyze the memory graph

// For more detailed memory analysis:
// 1. Open Memory tab
// 2. Take heap snapshot before interaction
// 3. Perform map operations
// 4. Take another heap snapshot
// 5. Compare snapshots to find memory leaks

// Code to help track DOM element counts
const countElements = () => {
  const counts = {
    markers: document.querySelectorAll('.leaflet-marker-icon').length,
    tiles: document.querySelectorAll('.leaflet-tile').length,
    containers: document.querySelectorAll('.leaflet-container').length,
    // Add other selectors as needed
  };

  console.table(counts);
  return counts;
};

// Call before and after operations to check for leaks
countElements();`}
        </pre>

        <h5>3. React Component Profiling</h5>
        <pre style={{ backgroundColor: '#f5f5f5', padding: '15px', borderRadius:
'4px', overflowX: 'auto' }}>
{`// Wrap components with React.memo to prevent unnecessary re-renders
const MapMarker = React.memo(({ position, name }) => {
  return (
    <Marker position={position}>
      <Popup>{name}</Popup>
    </Marker>
  );
});

// Use React DevTools Profiler:
// 1. Open React DevTools
// 2. Go to the Profiler tab
// 3. Click the record button
// 4. Interact with your map
// 5. Stop recording
// 6. Analyze which components are re-rendering and why`}
        </pre>
      </div>

      <div style={{ marginBottom: '25px' }}>
        <h4>Network Debugging for Tile Loading</h4>
```

```
        <h5>1. Monitor Tile Requests</h5>
        <p>
          Use the browser's network tab to analyze tile loading:
        </p>
        <ol style={{ marginBottom: '10px' }}>
          <li>Open browser DevTools and go to the Network tab</li>
          <li>Filter requests to show only image or XHR requests</li>
          <li>Look for patterns like "tile" or your tile server domain</li>
          <li>Check for 404 errors or slow-loading tiles</li>
          <li>Examine response headers for caching directives</li>
        </ol>

        <h5>2. Debug CORS Issues</h5>
        <pre style={{ backgroundColor: '#f5f5f5', padding: '15px', borderRadius:
'4px', overflowX: 'auto' }}>
{`// Common CORS error in console:
// "Access to image at 'https://tile-server.com/...' from origin
'http://localhost:3000'
// has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header..."

// Solutions:
// 1. Use a CORS-enabled tile server
// 2. Use a proxy server in development
// 3. Add crossOrigin option to TileLayer
<TileLayer
  url="https://tile-server.com/{z}/{x}/{y}.png"
  crossOrigin="anonymous" // Add this
/>

// 4. For development, you can use a proxy in package.json (Create React App)
// "proxy": "https://tile-server.com"`}
        </pre>

        <h5>3. Simulate Network Conditions</h5>
        <p>
          Test your map under different network conditions:
        </p>
        <ol>
          <li>In Chrome DevTools, go to the Network tab</li>
          <li>Use the throttling dropdown to simulate slow connections</li>
          <li>Try "Slow 3G" to test how your map behaves with slow tile
loading</li>
          <li>Add error handling for tile loading failures</li>
        </ol>
        <pre style={{ backgroundColor: '#f5f5f5', padding: '15px', borderRadius:
'4px', marginTop: '10px', overflowX: 'auto' }}>
{`// Add event listeners to track tile loading
const TileLoadingTracker = () => {
  const map = useMap();

  useEffect(() => {
    const tileEvents = {
      loading: () => console.log('Tiles loading...'),
      load: () => console.log('All tiles loaded'),
```

```
      tileerror: (e) => console.error('Tile error:', e)
    };

    Object.entries(tileEvents).forEach(([event, handler]) => {
      map.on(event, handler);
    });

    return () => {
      Object.entries(tileEvents).forEach(([event, handler]) => {
        map.off(event, handler);
      });
    };
  }, [map]);

  return null;
};`}
        </pre>
      </div>

      <div>
        <h4>General Debugging Tips</h4>

        <div style={{ marginBottom: '15px' }}>
          <h5>1. Use React DevTools for Component Inspection</h5>
          <ul>
            <li>Install the React DevTools browser extension</li>
            <li>Inspect component props and state</li>
            <li>Check for unnecessary re-renders</li>
            <li>Verify that refs are properly assigned</li>
          </ul>
        </div>

        <div style={{ marginBottom: '15px' }}>
          <h5>2. Isolate Issues with Minimal Examples</h5>
          <ul>
            <li>Create a simplified version of your map component</li>
            <li>Remove features one by one until the issue disappears</li>
            <li>Use tools like CodeSandbox or Stackblitz to create shareable
demos</li>
            <li>Test with different Leaflet and React-Leaflet versions</li>
          </ul>
        </div>

        <div style={{ marginBottom: '15px' }}>
          <h5>3. Check Component Lifecycle</h5>
          <pre style={{ backgroundColor: '#f5f5f5', padding: '15px', borderRadius:
'4px', overflowX: 'auto' }}>
{`// Add logging to track component lifecycle
useEffect(() => {
  console.log('Component mounted');
  return () => console.log('Component unmounted');
}, []);

useEffect(() => {
```

```
  console.log('Props or state changed:', { center, zoom });
}, [center, zoom]);`}
          </pre>
        </div>

        <div>
          <h5>4. Use Feature Flags for Troubleshooting</h5>
          <pre style={{ backgroundColor: '#f5f5f5', padding: '15px', borderRadius:
'4px', overflowX: 'auto' }}>
{`// Add feature flags to enable/disable specific features
const MapComponent = ({
  center,
  zoom,
  enableClustering = true,
  enableTooltips = true,
  enableRealTimeUpdates = true
}) => {
  return (
    <MapContainer center={center} zoom={zoom}>
      <TileLayer ... />

      {enableClustering ? (
        <MarkerClusterGroup>
          {markers.map(/* ... */)}
        </MarkerClusterGroup>
      ) : (
        <>{markers.map(/* ... */)}</>
      )}

      {enableTooltips && <CustomTooltips />}

      {enableRealTimeUpdates && <RealTimeUpdater />}
    </MapContainer>
  );
};`}
          </pre>
        </div>
      </div>
    </div>
  );
};

export default DebuggingTechniquesGuide;
```

Conclusion

Your journey with Leaflet.js and React is now well-equipped with comprehensive knowledge and practical examples. This guide covered everything from basic setup to advanced features, helping you create powerful, interactive maps in your React applications.

Key Takeaways

1. **Solid Foundation**: You've learned how to set up Leaflet with React, understand its core concepts, and build interactive maps that respond to user actions.
2. **Component Architecture**: The guide demonstrated how to create reusable, maintainable map components that follow React's best practices, including proper lifecycle management and state handling.
3. **Advanced Techniques**: From custom controls to clustering markers and visualizing complex datasets, you now have the tools to create sophisticated mapping solutions.
4. **Performance Optimization**: You've explored techniques to handle large datasets, optimize rendering, and create responsive maps that work well on all devices.
5. **Debugging and Maintenance**: The debugging section provided strategies to troubleshoot common issues and maintain your map components effectively.

Next Steps

As you continue your journey with Leaflet and React, consider exploring these areas:

- **Specialized Visualizations**: Explore advanced data visualizations like flow maps, choropleth maps, and animated time-series maps.
- **Integration with Backend Services**: Connect your maps to real-time data sources, GIS databases, or custom APIs.
- **Offline Mapping**: Implement offline support for mobile applications using technologies like IndexedDB to store map tiles.
- **Collaborative Features**: Add collaborative editing, shared views, or real-time collaboration features to your maps.
- **Accessibility Improvements**: Continue enhancing the accessibility of your map interfaces for users with disabilities.

Remember that the mapping ecosystem is constantly evolving. Stay updated with the latest developments in both Leaflet and React to take advantage of new features and improvements.

Good luck with your mapping projects! With the knowledge from this guide, you're well-prepared to create impressive, interactive map experiences in your React applications.