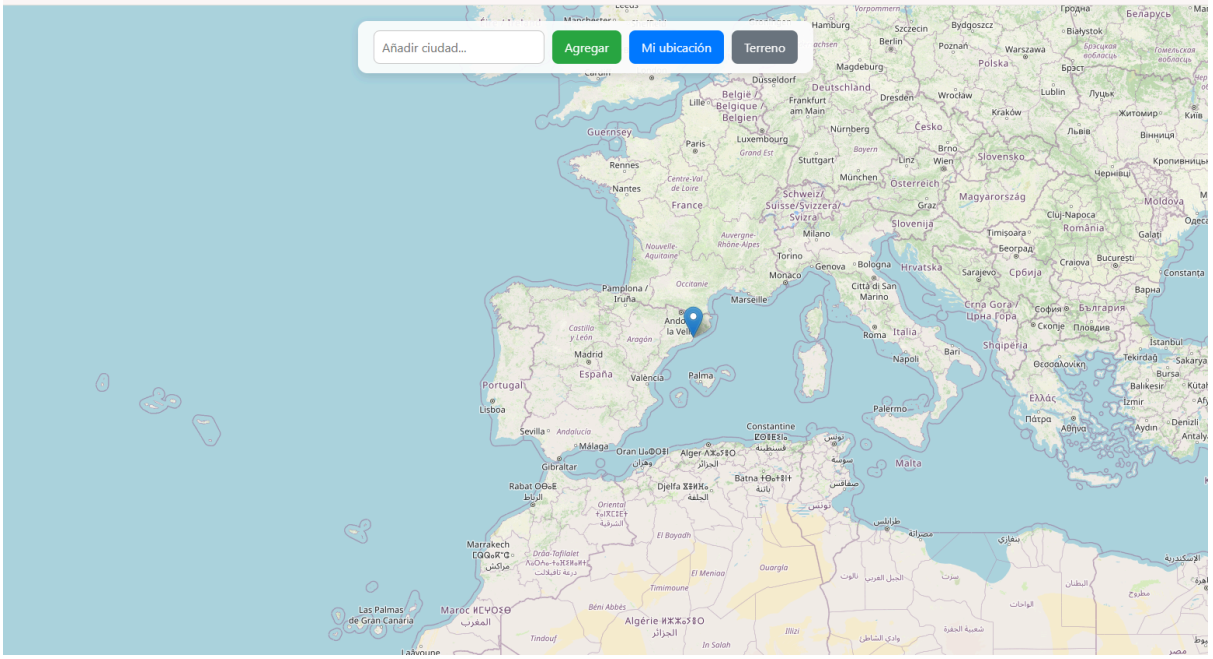
Documentación técnica : Aplicación de mapa con React Leaflet



github link : https://github.com/OthmanDouiri/react-map.git

Realizado por **Othman Douiri**

Indice:

1. Descripción general 3
2. Especificaciones técnicas 3
3. Estructura del código 4
    3.1 Componentes clave: 4
    3.2 Estructura del código: 4
    3.3 Funciones principales: 5
    3.4 Gestión de APIs 6
4. Uso, ejemplos y pruebas 6
    4.1 Buscar una ciudad: 6
    4.2 Ubicación actual: 8
    4.3 Cambiar estilo de mapa: 8
    4.4 Pruebas realizadas: 9
5.Posibles mejoras o extensiones 9

1. Descripción general

Esta aplicación es una herramienta interactiva basada en mapas, desarrollada con React y TypeScript, que permite al usuario:

Permite a los usuarios buscar ciudades, añadirlas al mapa, mostrar información meteorológica de las ubicaciones y cambiar el estilo visual del mapa.

El objetivo principal del proyecto es demostrar el uso combinado de librerías como **react-leaflet** con múltiples APIs externas, en este caso:

OpenCage Geocoder : para obtener coordenadas a partir del nombre de la ciudad)
OpenWeatherMap : para obtener información del clima), implementando una funcionalidad completa y útil para el usuario.

---

2. Especificaciones técnicas

- **Lenguaje:** TypeScript
- **Framework:** React 18 (Vite)
- **Librerías principales:**
    - react-leaflet : Para renderizar y gestionar el mapa interactivo.
    - Leaflet : Motor de mapas.
    - OpenCage Geocoder API : Para convertir nombres de ciudades en coordenadas geográficas.
    - OpenWeatherMap API : Para obtener datos meteorológicos.
- **Requisitos del sistema:**
    - Navegador moderno (Chrome, Firefox, Edge).
    - Conexión a internet (para acceder a las APIs y tiles del mapa).

---

3. Estructura del código

3.1 Componentes clave:

- **MapView.tsx:** Componente principal que gestiona el mapa, la lógica de búsqueda, la geolocalización, los marcadores y el cambio de estilo del mapa.

3.2 Estructura del código:

Definición de tipos :

```
type City = {
  id: number;
  name: string;
  lat: number;
  lng: number;
  weather: string | null;
};
```

Define la estructura de datos para las ciudades.
Cada ciudad tiene un ID único, nombre, coordenadas (latitud y longitud) e información del clima.

3.3 Funciones principales:

- **addCity()**: Busca una ciudad por nombre (usando OpenCage), obtiene sus coordenadas y clima (OpenWeather) y añade un marcador al mapa.

```
// Llamada a la API de geolocalización con OpenCage
const apiKey = import.meta.env.VITE_OPENCAGE_API_KEY;
const res = await fetch(
  `https://api.opencagedata.com/geocode/v1/json?q=${encodeURIComponent(query)}&key=${apiKey}`
);
const data = await res.json();
```

Aquí se realiza una llamada a la API de **OpenCage** usando el nombre de la ciudad que el usuario ha ingresado (query). El encodeURIComponent(query) asegura que la ciudad se formatee correctamente en la URL. La clave API (VITE_OPENCAGE_API_KEY) se obtiene de las variables de entorno. La respuesta se convierte en formato JSON y se guarda en la variable data.

```
const { lat, lng } = data.results[0].geometry;
const label = data.results[0].formatted;
```

Aquí se extraen la **latitud** (lat) y **longitud** (lng) de la primera ciudad encontrada en los resultados. Además, se obtiene el nombre completo de la ciudad (label), que es la propiedad formatted de la respuesta.

```
const weatherKey = import.meta.env.VITE_OPENWEATHER_API_KEY;
const weatherRes = await fetch(
  `https://api.openweathermap.org/data/2.5/weather?lat=${lat}&lon=${lng}&units=metric&appid=${weatherKey}&lang=es`
);
const weatherData = await weatherRes.json();
```

Esta sección realiza una llamada a la API de **OpenWeather** usando las coordenadas (lat, lng) obtenidas previamente para obtener el clima de la ciudad. La respuesta se guarda en weatherData.

```
const weather =
  weatherData?.weather?.[0]?.description && weatherData?.main?.temp
    ? `${weatherData.weather[0].description} · ${weatherData.main.temp}°C`
    : null;
```

Aquí se extrae la descripción del clima (weatherData.weather[0].description) y la temperatura en grados Celsius (weatherData.main.temp). Si ambos datos están disponibles, se formatean en un string que contiene la descripción del clima y la temperatura (por ejemplo, "Despejado · 25°C"). Si no se encuentra información del clima, se asigna null.

```
// Nueva ciudad a agregar
const newCity: City = {
  id: Date.now(),
  name: label,
  lat,
  lng,
  weather,
};
```

Se crea un nuevo objeto City que contiene la información de la ciudad: un id único (usando Date.now() para asegurar que sea único), el nombre de la ciudad (name), las coordenadas (lat, lng), y el clima (weather).

```
// Actualiza el estado con la nueva ciudad
setCities((prev) => [...prev, newCity]);
setQuery('');
mapRef.current?.setView([lat, lng], 10);
```

Aquí se actualiza el estado cities agregando la nueva ciudad a la lista existente de ciudades. El método setCities recibe una función que añade la ciudad al final de la lista (...prev representa las ciudades anteriores). También se limpia el campo de búsqueda (setQuery('')).

Y finalmente, el mapa se centra en la nueva ciudad utilizando las coordenadas lat y lng y se ajusta el zoom a nivel 10.

- **removeCity(id)**: Elimina una ciudad del array y por tanto su marcador.

```
const removeCity = (id: number) => {
  setCities((prev) => prev.filter((city) => city.id !== id));
};
```

- **locateMe()**: Usa la API de geolocalización del navegador para centrar el mapa en la posición actual del usuario, también usa la api de openweathermap para obtener los datos meteorológico

```
// Obtener la ubicación actual del usuario
navigator.geolocation.getCurrentPosition(async (pos) => {
  const { latitude, longitude } = pos.coords;
  mapRef.current?.setView([latitude, longitude], 10);
```

Aquí si la geolocalización está disponible, se obtiene la posición actual del usuario con navigator.geolocation.getCurrentPosition. La información de la ubicación (latitude y longitude) se extrae de la respuesta (pos.coords).

```
setCities((prevCities) => {
  const exists = prevCities.find((city) => city.name === 'Mi ubicación');
  if (exists) {
    // Si ya existe, actualizar posición y clima
    return prevCities.map((city) =>
      city.name === 'Mi ubicación' ? { ...city, lat: latitude, lng: longitude, weather }: city
    );
  } else {
    // Si no existe, agregarla
    return [
      ...prevCities,
      {
        id: Date.now(),
        name: 'Mi ubicación',
        lat: latitude,
        lng: longitude,
        weather,
      },
    ];
  }
});
```

Aquí se verifica si ya existe una ciudad llamada "Mi ubicación" en el estado cities. Si ya existe, se actualiza con las nuevas coordenadas y el clima. Si no existe, se agrega una nueva entrada con el nombre "Mi ubicación", las coordenadas y el clima correspondiente.

- **getTileLayer():** Devuelve la URL del estilo de mapa seleccionado (calles o terreno).

```
const getTileLayer = () => {
  return mapStyle === 'streets'
    ? 'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png'
    : 'https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png';
};
```

Gestión de APIs

- Ambas APIs están protegidas con variables de entorno (import.meta.env.VITE_*).

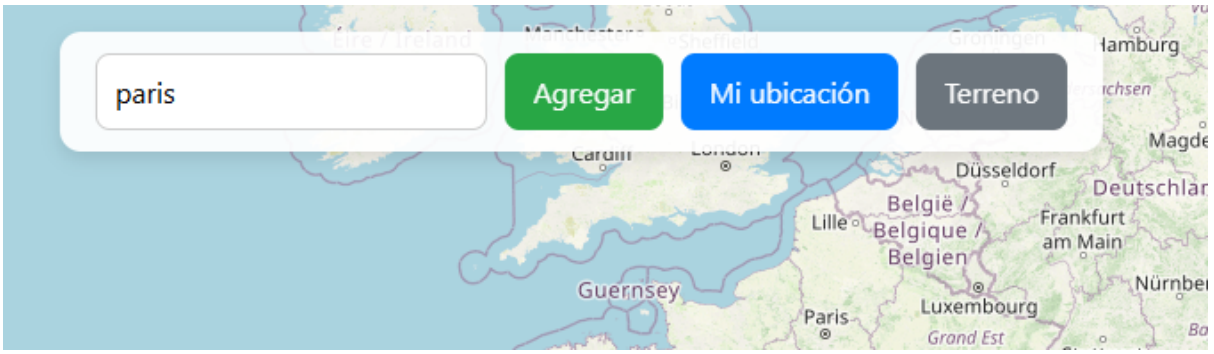
```
.env
1  VITE_OPENCAGE_API_KEY=2bcfff06715041a895db57a54ad67157
2  VITE_OPENWEATHER_API_KEY=722c4a4264b65bd8848e0037c959e765
3
4
```

---

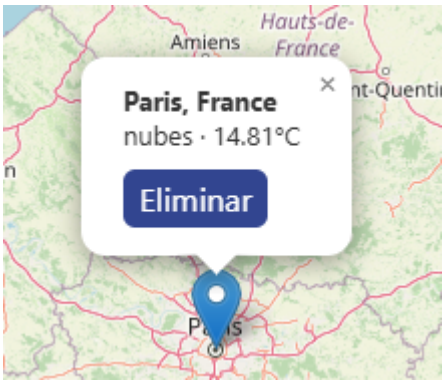
4. Uso, ejemplos y pruebas

4.1 Buscar una ciudad:

1. Introduce una ciudad en el input (por ejemplo, Paris).

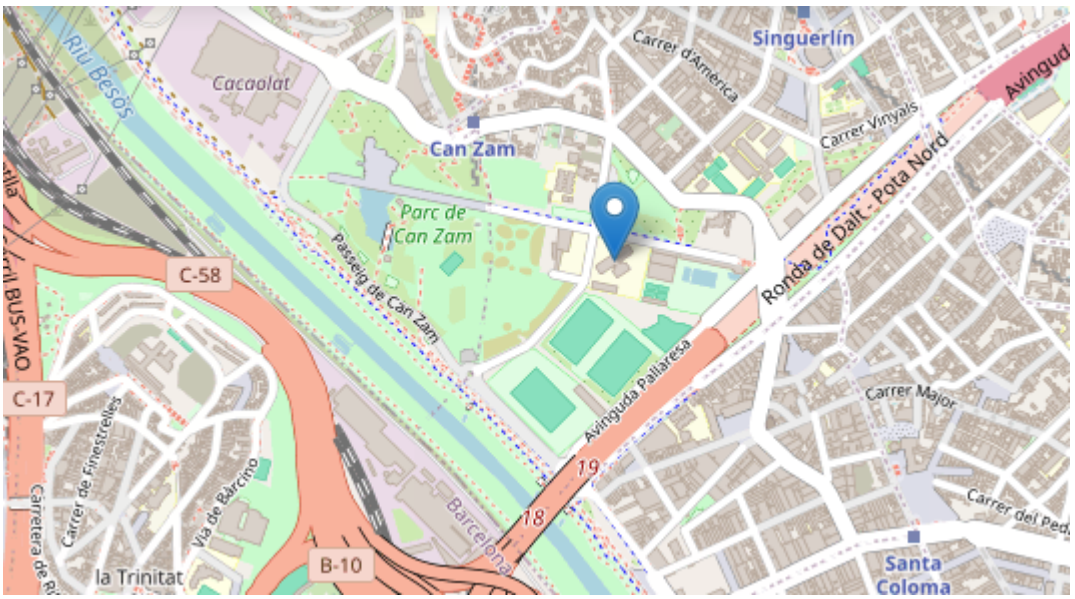


2. Pulsa el botón "Agregar".
3. Se coloca un marcador en el mapa, mostrando su nombre y clima actual.
4. En el popup puedes eliminar la ciudad.



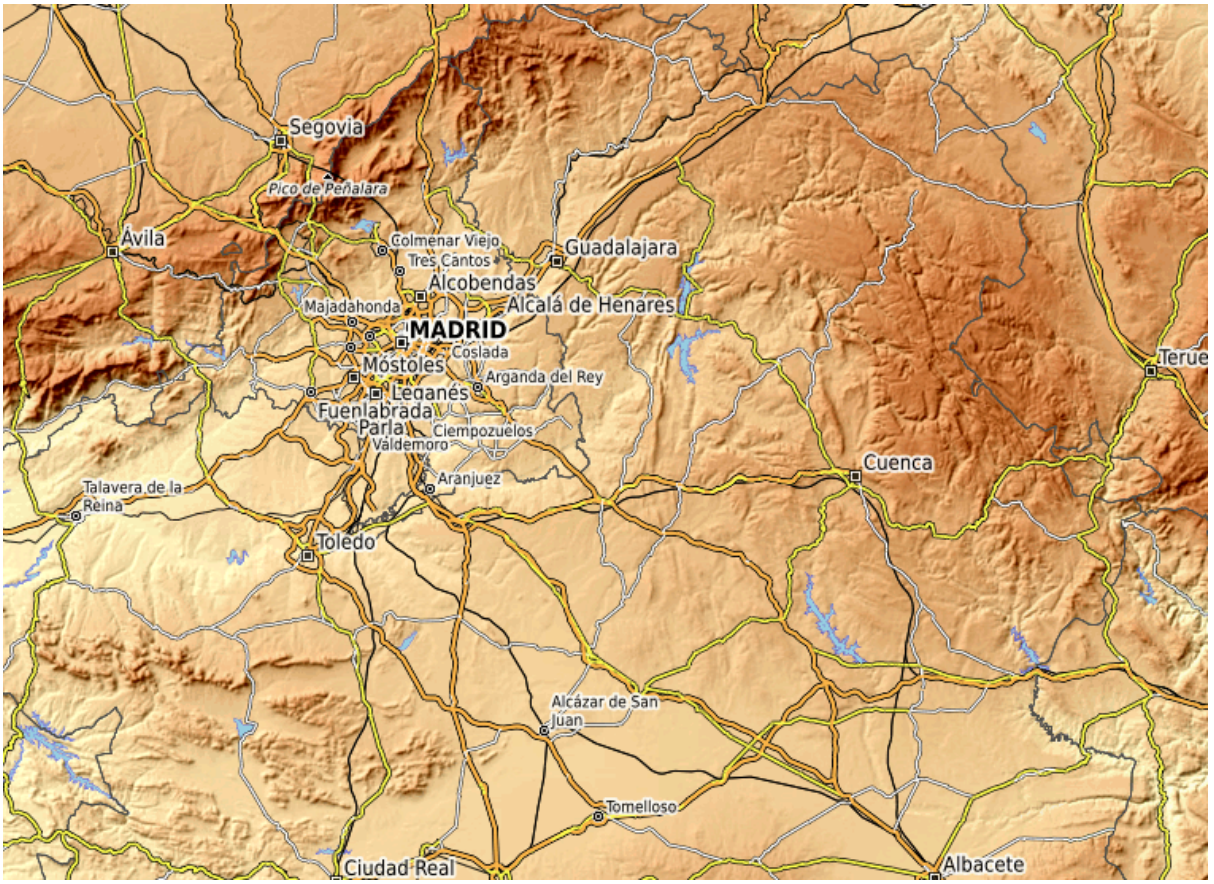
4.2 Ubicación actual:

- Pulsa el botón “Mi ubicación” y se centra el mapa automáticamente sobre tu posición.



4.3 Cambiar estilo de mapa:

- Pulsa el botón "Calles"/"Terreno" para alternar entre los dos estilos.



4.4 Pruebas realizadas:

- Búsqueda de ciudades conocidas y desconocidas.
- Eliminación de ciudades.
- Pruebas con conexión lenta/desconectada.
- Pruebas con y sin permisos de geolocalización.

5.Posibles mejoras o extensiones

- Guardar las ciudades en localStorage para mantenerlas entre sesiones.
- Autocompletado del input al escribir ciudades (usando por ejemplo la API de Mapbox o Algolia Places).
- Mostrar más datos del clima.
- Historial de ciudades buscadas.
- Modo oscuro.