App Meteo

Relazione finale

1) Introduzione

Il progetto sviluppato è una applicazione web **lato client** realizzata in **JavaScript** che integra dati geografici e meteorologici attraverso l'uso di **servizi HTTP esterni** e di librerie dedicate alla visualizzazione su **mappa**. L'applicazione consente di individuare località italiane tramite l'utilizzo di select, visualizzarle su una mappa interattiva e ottenere informazioni meteo aggiornate. Inoltre per ogni località italiana permette di visualizzare una **pagina di dettaglio**, con informazioni meteo più dettagliate fino a **16 giorni**. In aggiunta è possibile selezionare una località preferita, la quale rimarrà **memorizzata** anche alla chiusura del browser, tramite l'uso di **Local Storage**.

Dal punto di vista tecnico, il progetto fa uso di **array JavaScript** e dei relativi metodi per la gestione e la selezione dei dati provenienti da file esterni (come l'elenco dei comuni italiani). Per la rappresentazione geografica viene utilizzata la libreria **Leaflet**, basata su **OpenStreetMap**, che permette la creazione della mappa, l'aggiunta di livelli (tile layer) e la gestione dei marker.
Il recupero dei dati avviene tramite **chiamate HTTP asincrone** effettuate con fetch, in particolare verso il servizio **Open Meteo**, gestendo le risposte in formato JSON.

Nella relazione verranno illustrati i principali metodi JavaScript utilizzati, le funzionalità fondamentali di Leaflet/OpenStreetMap e la logica delle chiamate ai servizi esterni, supportando la spiegazione con **estratti di codice** e **screenshot** dell'applicazione.

2) Metodi JavaScript per la manipolazione degli array.

Per la manipolazione degli array abbiano utilizzato 4 metodi principali.

1. Filter

Filter è un metodo degli **array** che prende come paramentro una **callback**. Essa a sua volta ha come parametro il **singolo elemento** dell'array. Essa restituisce un valore **booleano**. Filter esegue questa callback tante volte quanto il **numero di elementi** presenti nel array, ogni volta cambiando l'elemento passato alla callback. Filter restituisce un **array** contenete i soli elementi che passati alla callback, essa ha restituito **True**.

Nella figura (2.a) **tuttiIComuni** è l'array non filtrato, al quale viene

```
const arrFiltrato=tuttiIComuni.
                  filter(function(x){
                      return x.regione.nome==regione;
                  })
```

Figura 2.A

applicato filter. La **callback** restitusce il risultato del confronto tra il nome della regione del **comune** (x) e la regione selezionata dal utente (**regione**). Il risultato di filter viene assegnato a **arrFiltrato**.

2. Map

Map è un metodo degli **array** che prende come paramentro una **callback**. Essa a sua volta ha come parametro il **singolo elemento** dell'array. La callback modifica e restituisce il valore del elemento passato. Map esegue questa callback tante volte quanto il **numero di elementi** presenti nel array, ogni volta cambiando l'elemento passato alla callback. Map restituisce un **array** contenete i valori modificati degli elementi passati alla callback.

Nella figura (2.b) **arrFiltrato** è l'array originale, al quale viene applicato map. La **callback** restitusce il nome della provincia del **comune** (x). Il risultato di map viene assegnato a **arrayProvincie**.

```
const arrayProvincie= arrFiltrato.map(function(x){
                        return x.provincia.nome;
                    })
```

Figura 2.B

3. Spread operator e push

Per aggiungere più elementi a un array è stato utilizzato il metodo push() insieme allo **spread operator (...)**, che permette di espandere un array in una lista di valori singoli. In questo modo gli elementi vengono inseriti direttamente nell'array di destinazione senza c reare array annidati.

```
regioni.push(...tuttiIComuni.map(function(x){
    return x.regione.nome;
}))
```

Figura 2.C

Nella figura (2.c) **tuttiIComuni** è l'array originale, al quale viene applicato map, il quale risultato (come abbiamo tratto nel punto 1.2) è un **array**. Se lo inserissimo senza lo spread operator avremmo un array **bidimensionale**. Invece in questo modo vengo abbiamo un array **monodimesionale**.

4. Sort

Il metodo sort() di JavaScript viene utilizzato per **ordinare gli elementi di un array**. Di default ordina gli elementi come **stringhe in ordine alfabetico**, anche se si tratta di numeri, quindi spesso è necessario fornire una **funzione di confronto** (callback) per ottenere un ordinamento corretto.

Nella figura (2.d) non è stato necessario usare un callback, poiché **arrayRegioni** contiene stringhe.

```
arrayRegioni.sort()
```

Figura 2.D

3) Metodi Leaflet per l'utilizzo della mappa.

Per visualizzare e gestire le mappe abbiamo utilizzato la libreria **Leaflet**, basata su **OpenStreetMap**. Nella nostra applicazione sono stati impiegati diversi **metodi e oggetti principali**, tra cui:

1. L.map

Rappresenta l'oggetto mappa principale, sul quale vengono aggiunti marker, layer e popup. Inizialmente viene creato e collegato a un elemento HTML.

2. Metodi per il controllo della visuale

- **setView([lat, lon], zoom)** Esso imposta il centro della mappa e il livello di zoom.
- **flyTo([lat, lon], zoom)** Esso sposta la mappa con un'animazione fluida verso un punto specifico.
- **fitBounds(bounds)** Esso adatta automaticamente la visuale per mostrare un insieme di punti (o marker) all'interno della mappa.

Questi metodi sono stati utilizzati per **centrare la mappa sulla località selezionata** dall'utente o per adattare la visuale a più marker contemporaneamente.

3. Oggetti principali

- **L.marker([lat, lon])** Esso rappresenta un marcatore sulla mappa.
- **L.popup()** Esso visualizza informazioni aggiuntive quando l'utente clicca sul marker o sulla mappa.
- **L.layerGroup([...])** Esso raggruppa più marker o layer, permettendo di aggiungerli o rimuoverli contemporaneamente dalla mappa.

Nella figura 3.a si può vedere un esempio di come creare un mappa e come usare setView.

```
const mappa = L.map('mappa').setView([41.9, 12.483333 ], 6);
const gruppo=L.layerGroup().addTo(mappa);
```

Figura 3.A

Inoltre si vede anche come creare un layerGroup. Per ogni mappa bisogna aggiugere un tile layer. Nella figura 3.b si vede come aggiungere il tile layer di **OpenStreetMap.**

```
L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
}).addTo(mappa);
```

Figura 3.B

Nella figura 3.c si vede come creare un popup ed aggiugerlo ad un marker.

Nella figura 3.d si vede un utilizzo di fitbounds.

```
marker.bindPopup(contenutoPopup).openPopup();

marker.addTo(gruppo)
```

Figure 3.C e 3.D

```
mappa.fitBounds([[minLat, minLon],[maxLat, maxLon]]);
```

4) Logica delle chiamate ai servizi HTTP

Nell'applicazione i dati esterni vengono caricati tramite il metodo **fetch**(), che permette di inviare richieste **HTTP** e ricevere le risposte in formato JSON. Una volta ricevuto il file JSON, esso viene trasformato in un array tramite l'uso del metodo .**json**(). Inoltre in caso di **indisponibilità** del server o nel caso di **connessione assente** viene sollevato un **errore**, il quale viene catturato e visualizzato al utente. Il codice relativo è riportato nelle figure 4.a e 4.b.

```
async function ricevi() {
    const URL_COMUNI="https://raw.githubusercontent.com/matteocontrini/comuni-json/master/comuni.json";
    const rispostaServer= await fetch(URL_COMUNI);

    if (!rispostaServer.ok){
        throw new Error ("Errore di rete");
    }

    const arrayJSON=await rispostaServer.json();

    return arrayJSON;
}
```

Figura 4.A

```
ricevi()
  .then(risposta => {
      for(let i=0; i<risposta.length; i++){
          tuttiIComuni.push(risposta[i])
      }

      clearInterval(caricamento);
      paragrafoCaricamento.innerHTML=""

      creaRegioni();
  })
  .catch(() => {
    clearInterval(caricamento);
    paragrafo.textContent =
      "Errore nel caricamento dei comuni. Riprovare più tardi.";
  });
```

Figura 4.B

Le informazioni meteorologiche, invece, sono ottenute direttamente dal servizio **Open Meteo**. Grazie all'uso di **async/await**, l'applicazione attende che le risposte siano completamente ricevute e convertite in oggetti JavaScript prima di aggiornare l'interfaccia. Una volta disponibili, questi dati vengono integrati nella mappa attraverso **marker e popup**, oppure visualizzati nelle card dell'interfaccia utente, permettendo all'utente di consultare informazioni aggiornate in modo chiaro e immediato. Questo approccio garantisce che la mappa e le informazioni visualizzate rimangano sempre sincronizzate con la scelta effettuata dall'utente. Il relativo codice è riportato nella immagini 4.c e 4.d.

```
async function meteoComune(lan, lon) {
    const URL_METEO="https://api.open-meteo.com/v1/forecast?"

    const URL_MODIFICATO=URL_METEO+"latitude="+lan+"&longitude="+lon+"&current=temperature_2m"

    const rispostaServer= await fetch(URL_MODIFICATO)

    if (!rispostaServer.ok) return "Si è verificato un errore di rete"

    const risposta_meteo=await rispostaServer.json();

    return risposta_meteo.current.temperature_2m
}
```

Figure 4.C e 4.D

```
    meteoComune(risultato[0], risultato[1]).then(risposta =>{
        contenutoPopup+="<br>Temperatura Attuale: "+risposta+"°C"

        contenutoPopup+='<br><button type="button" class="btn btn-primary"><a href="'+generaLinkDettaglio(comune.nome, risultato[0], risultato[1

        marker.bindPopup(contenutoPopup).openPopup();

        marker.addTo(gruppo)

    })
```

5) Conclusione

Il progetto ha permesso di realizzare un'applicazione web lato client in grado di visualizzare informazioni geografiche e meteorologiche in modo interattivo. Grazie all'uso dei metodi JavaScript sugli **array**, siamo stati in grado di **filtrare** e trasformare i dati dei comuni italiani secondo le **scelte dell'utente**. L'integrazione della libreria **Leaflet** e di **OpenStreetMap** ha consentito di creare una **mappa** interattiva con **marker**, **popup** e **layer**, rendendo immediata la consultazione dei dati. Le chiamate ai servizi esterni, gestite con **fetch**() e **async/await**, hanno garantito che le informazioni meteo fossero sempre aggiornate. Questo progetto ci ha permesso di comprendere meglio la manipolazione dei dati, la gestione di oggetti complessi e l'interazione con **API esterne**, applicando in modo pratico concetti fondamentali di JavaScript e sviluppo web.