



# DELFRATE RICCARDO

Corso di Programmazione Web e Mobile

A.A 2021/2022

## Sunny.it

### Sommario

Sunny.it .....	1
Introduzione .....	2
Analisi dei requisiti .....	2
Struttura organizzativa .....	3
Work Breakdown Structure .....	3
Diagramma di Gantt.....	3
Flusso dei dati .....	4
Interfacce .....	5
Implementazione .....	8
Risorse.....	8
Schema .....	9
Server .....	10
DarkMode .....	11
AJAX .....	12
Api.....	13
Immagini .....	15
Grafici .....	17
Mappa.....	18
Sitografia .....	21

# Introduzione

Sunny.it è un'applicazione web responsive multiplatforma volta alla realizzazione di un servizio meteorologico sulle più grandi città mondiali.

All'interno dell'applicazione viene data all'utente la possibilità di scegliere la città di cui desidera vedere le informazioni meteorologiche oppure navigare per curiosità tra le città suggerite.

Sfrutta la tecnologia API con richieste al servizio [OpenWeather](#) per ottenere le informazioni meteo e richieste al servizio [unsplash](#) per ottenere immagini della città desiderata.

## Analisi dei requisiti

### Destinatari

Sunny.it è destinata a qualsiasi utente con o senza esperienza data la sua semplicità e la sua interfaccia user friendly.

Essendo un servizio situato su un server web è accessibile da qualsiasi parte del mondo avendo una connessione alla rete internet.

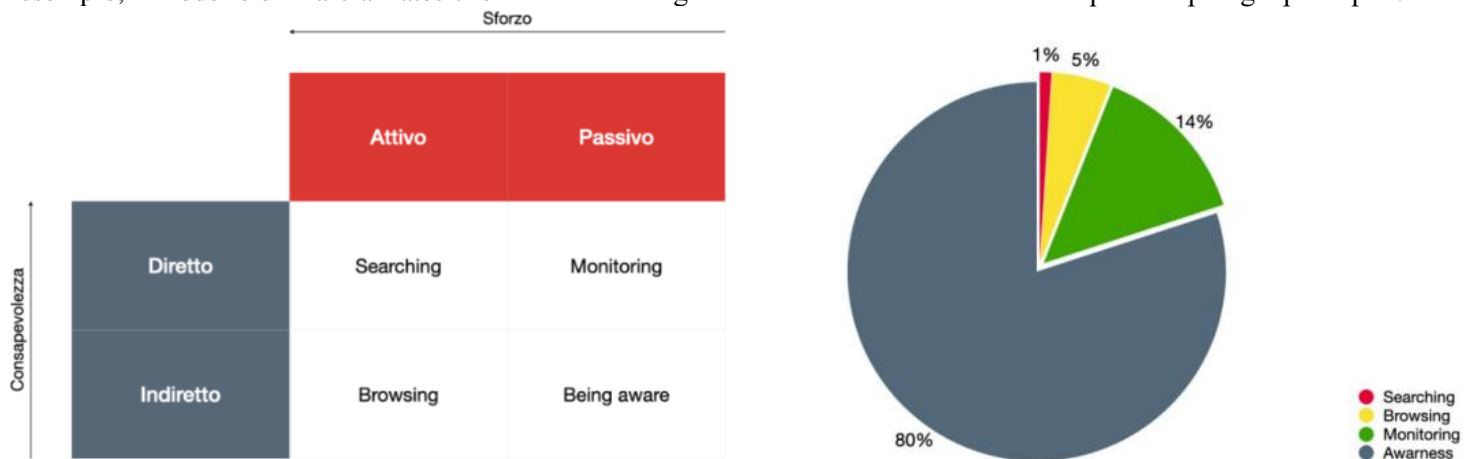
Permette in oltre la possibilità di essere visionato su qualsiasi dispositivo in quanto in fase di progettazione è stato utilizzato un design responsive ovvero un design adattabile alle dimensioni dello schermo.

### Modello di valore

Sebbene esistano già diversi siti o applicazioni web che offrono lo stesso servizio, Sunny.it si differenzia grazie alla sua interfaccia minimalista ma con tutte le informazioni necessarie evitando odiosi alert a comparsa e pubblicità che possono destabilizzare un utente alla ricerca delle informazioni.

Offre in oltre la possibilità di scegliere tra modalità giorno o notte, la quale andrà ad attenuare i colori per un maggior confort notturno o se si dispone di un telefono con tecnologia AMOLED od OLED permetterà un minor consumo della batteria.

È importante organizzare i contenuti in modo che le diverse tipologie di utenti trovino il loro percorso, sfruttando, ad esempio, il modello di Marcia Bates che divide le strategie di ricerca dell'informazione in quattro tipologie principali:



### Searching

Si effettua una ricerca mirata consapevole per cercare il meteo di una città.

### Monitoring

Monitoro la situazione meteorologica in maniera giornaliera senza uno scopo mirato.

### Browsing

Esploro senza avere un interesse specifico o un bisogno conoscitivo ma mi espongo attivamente alla possibilità di acquisire nuove informazioni

### Awareness

Sono le informazioni a venirci incontro, divento consapevole di qualcosa senza cercarla ad esempio tramite pubblicità.

### Usabilità

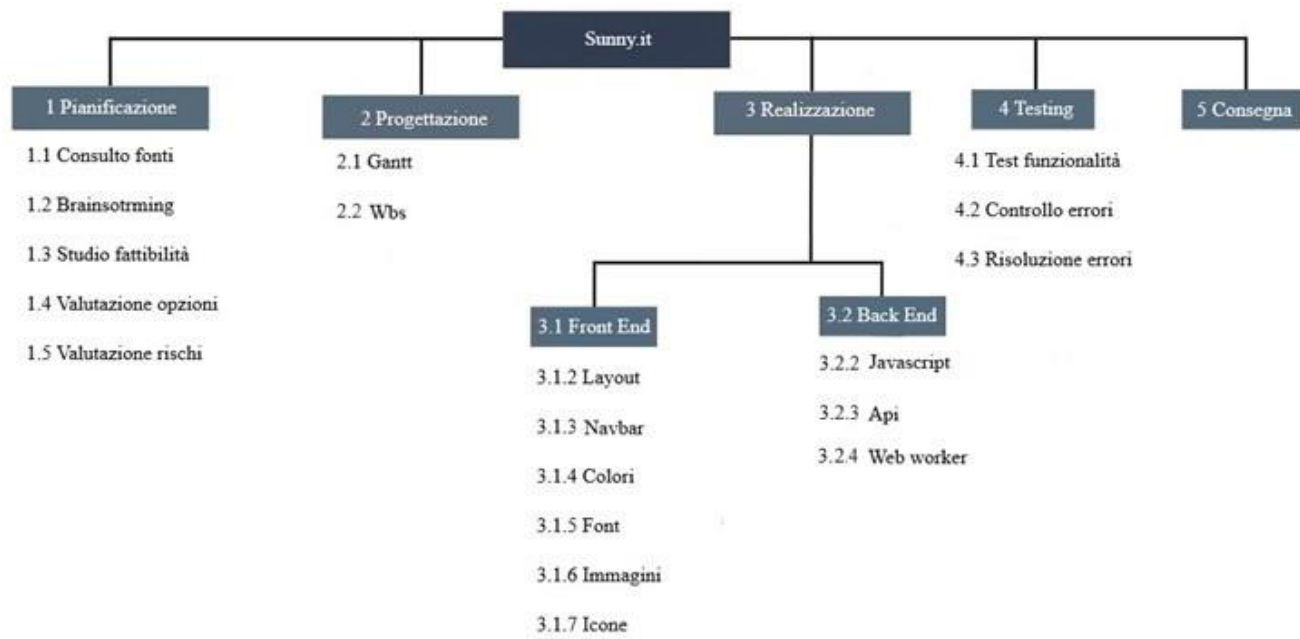
Per favorire l'usabilità utente dell'applicazione, sono stati resi visibili solo gli elementi utilizzabili e le azioni che si possono svolgere sono state guidate in modo da rendere il sistema il più naturale e familiare possibile.

Per ovviare ai problemi di refresh della schermata i quali comportano una destabilizzazione a livello visivo nell'utente si è optato per una struttura "Templating" nella quale vengono aggiornate solo le informazioni fondamentali, nel nostro caso quelle meteorologiche lasciando lo sfondo statico.

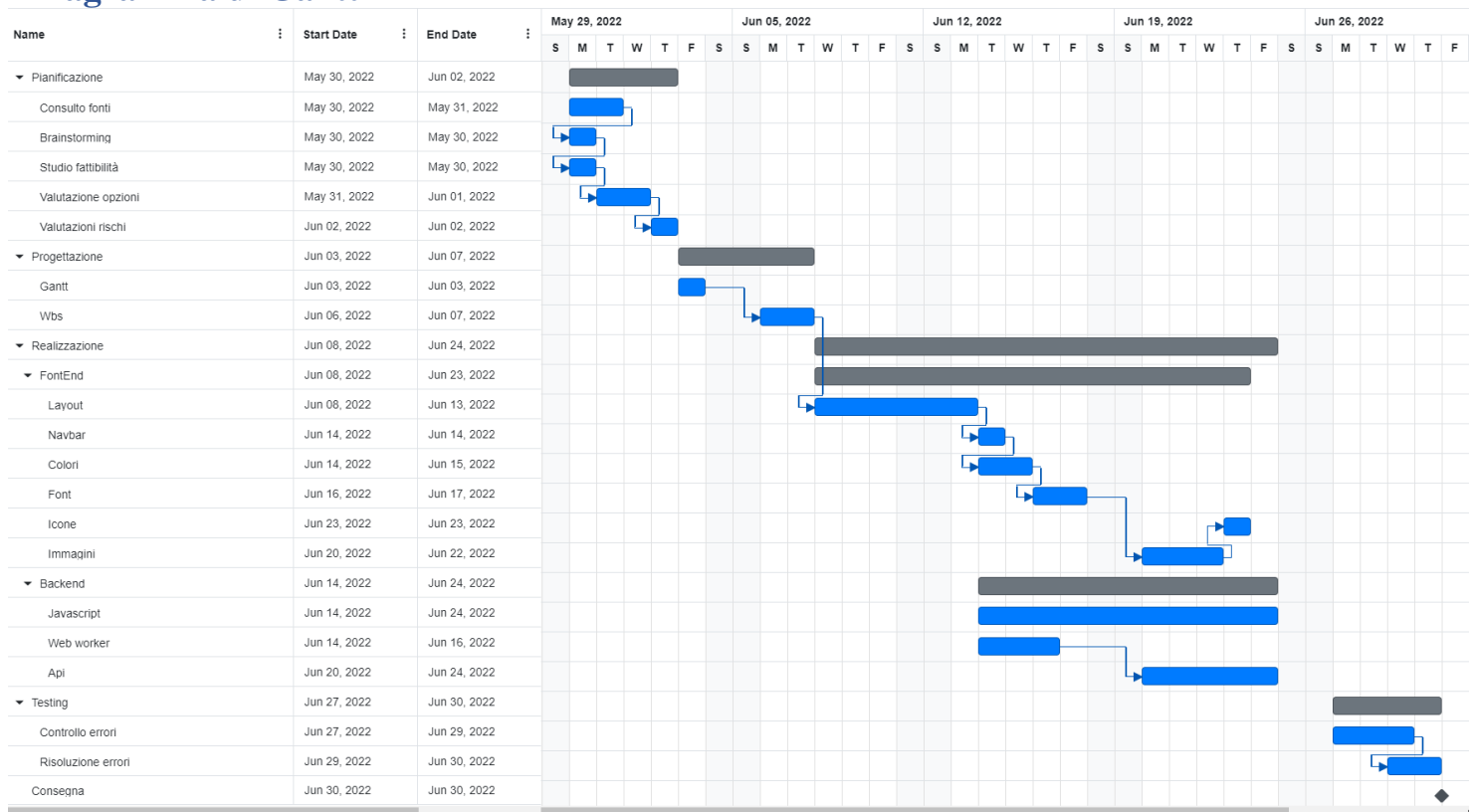
Come elemento finale ma non per importanza è stato creato un ciclo di vita separato tra dati e interfaccia, separando logicamente la struttura lo stile e il contenuto.

# Struttura organizzativa

## Work Breakdown Structure



## Diagramma di Gantt



## Flusso dei dati

I contenuti sono stati reperiti da internet prestando attenzione all'assenza di diritto d'autore, i costi di produzione e riadattamento sono nulli.

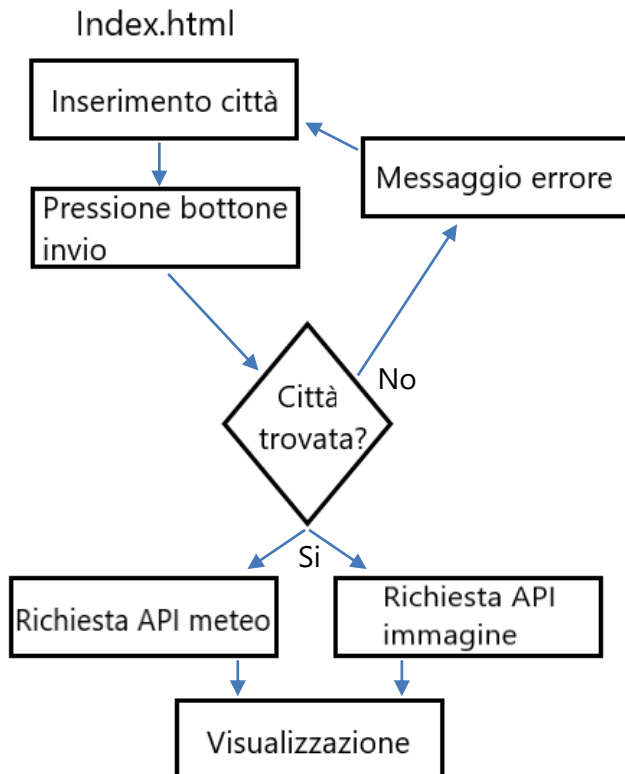
Tutte le immagini, tranne la favicon, sono reperite tramite API in modo tale da non avere una archiviazione locale.

Le previsioni meteo vengono aggiornate ad ogni accesso tramite richiesta al servizio OpenWeather.

Il contenuto deve essere percepito come d'interesse per i destinatari ed è stato quindi espresso in uno stile adeguato, richiedendo il minimo sforzo di attenzione da parte del destinatario e garantendo il massimo dell'informazione ricercata da quest'ultimo. I dati sono elementi indipendenti dalla struttura della pagina e hanno quindi un ciclo di vita indipendente e separato. L'unico punto di scambio dati tra browser e server avviene nel momento della ricerca della città desiderata, dove viene richiesto al server il luogo cercato.

### Flusso index.html

Di seguito viene rappresentato il flusso dei dati all'interno della pagina index.html



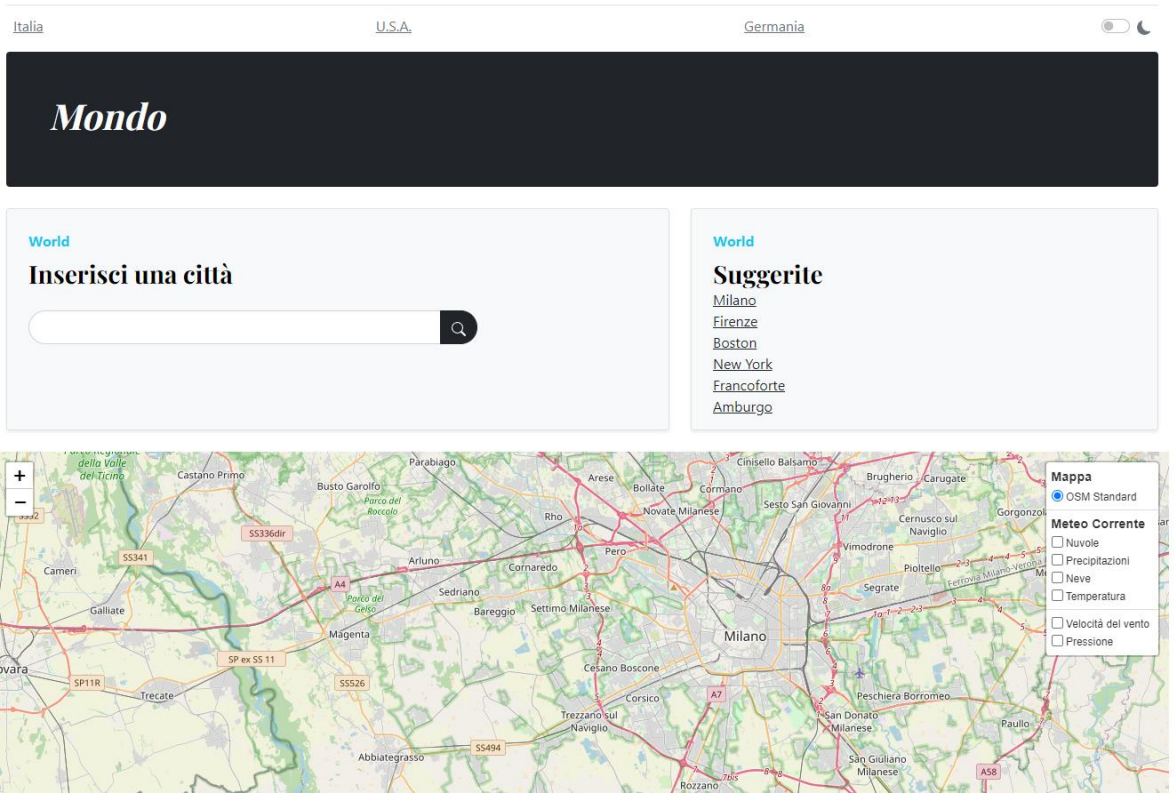
# Interfacce

Il layout è strutturato su un'unica pagina con riadattamenti per dispositivi mobile.

## Desktop



Sunny.it

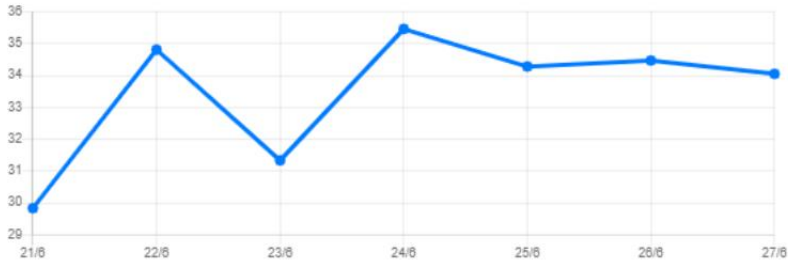


Nome del sito			
Link	Link	Link	DarkMode
Banner nazione			
Nome capitale		Immagine	
Info meteo			
Grafico			
Nome città		Nome città	
Info meteo		Info meteo	
Immagine		Immagine	

Italia

Roma

Temperatura: 29.83°  
Percepita: 28.6°  
Pressione: 1015 hPa  
Umidità: 30%  
Vento: 1.54 m/sec



Italia

Milano

Temperatura: 27.87°  
Percepita: 28.95°



Italia

Firenze

Temperatura: 28.33°  
Percepita: 29.36°





Sunny.it

ItaliaU.S.A.Germania

Mondo

World

Inserisci una città

World

Suggerite

Milano

Firenze

Boston

New York

Francoforte

Amburgo

+

-

Mappa

OSM Standard

Meteo Corrente

☐ Nuvole

☐ Precipitazioni

☐ Neve

☐ Temperatura

Italia

Roma

Temperatura: 27.18°

Percepita: 28.81°

Pressione: 1015 hPa

Umidità: 66%

Vento: 2.06 m/sec

34

33

32

31

30

29

28

27

31/8

32/8

33/8

34/8

35/8

36/8

37/8

Italia

Milano

Temperatura: 21°

Percepita: 21.43°

Italia


Firenze

Temperatura: 24.85°

Percepita: 25.69°

United states of America

Washington





# Implementazione

## Risorse

C:.	
.env	//Porta su cui si avvia il server
package-lock.json	//Informazioni sulle dipendenze del progetto
package.json	//Informazioni sulle dipendenze del progetto
server.js	//Server
assets	
css	
blog.css	//Style bootstrap base <i>importato</i>
bootstrap-dark.css	//Style dark mode <i>importato</i>
bootstrap-dark.css.map	//Style dark mode <i>importato</i>
bootstrap.min.css	//Style normal mode <i>importato</i>
bootstrap.min.css.map	//Style normal mode <i>importato</i>
leaflet.css	//Style mappa <i>importato</i>
map.css	//Style mappa <i>importato</i>
style.css	//Style
img	
favicon.png	
js	
bootstrap.bundle.min.js	//Js template <i>importato</i>
bootstrap.bundle.min.js.map	//Js template <i>importato</i>
css_switch.js	//Scambio normale/dark mode
getMeteo.js	//Api per ottenere info meteo
leaflet-openweathermap.js	//Gestione mappa <i>importato</i>
leaflet.js	//Gestione mappa <i>importato</i>
map.js	//Gestione mappa <i>importato</i>
rememberDark.js	//Local storage per dark mode
search_world.js	//Ricerca città globale
webWorker.js	//Web worker immagini
json	
citta.json	//Json contenente le città statiche
schema.json	//Schema per la validazione del sito
node_modules	//Moduli nodeJs <i>importato</i>
views	
index.html	//Index



## Schema

Tramite il seguente file Json ho specificato lo schema che l'applicazione web seguirà.

Fonte: [schema.org](https://schema.org)

```
{
  "@type": "WebPage",
  "@context": "https://schema.org",
  "name": "Sunny.it",
  "description": "Sito online per il meteo",
  "author": "Riccardo Delfrate 987362",
  "mainEntity": {
    "@type": "WebPage",
    "mainContentOfPage": {
      "@type": "WebPageElement",
      "name": "Italia U.S.A. Germania"
    },
    "contentLocation": [
      {
        "@type": "Place",
        "name": "Italia"
      },
      {
        "@type": "Place",
        "name": "USA"
      },
      {
        "@type": "Place",
        "name": "Deutschland"
      }
    ]
  },
  "publisher": {
    "@type": "CollegeOrUniversity",
    "name": "Unimi"
  },
  "editor": {
    "@type": "Person",
    "name": "Riccardo Delfrate 987362 Unimi: Sicurezza dei sistemi e delle reti informatiche"
  },
  "copyrightYear": "2022"
}
```

## Server

L'applicazione web è resa disponibile tramite un server realizzato in NodeJs strutturato nel seguente modo:

```
const express = require('express');
const path = require('path');
require("dotenv").config();
const app = express();
const port = process.env.PORT || 8000;

app.use(express.static(__dirname + '/assets'));

app.get('/', function (req, res) {
  res.sendFile(path.join(__dirname, '/views/index.html'));
});

app.get('/index', function (req, res) {
  res.sendFile(path.join(__dirname, '/views/index.html'));
});

app.listen(port);
console.log('Server started at http://localhost:' + port);
```

Dopo aver importato i moduli necessari specifico che tipo di richieste URL il server possa ricevere e che risposta debba dare.

## DarkMode

Attraverso un event listener applicato allo switch (checkbox modificata tramite bootstrap) catturo l'evento e sostituisco il css in modo da cambiare l'aspetto dell'applicazione web.

Successivamente salvo lo stato nel local storage in modo da averlo disponibile al caricamento successivo della pagina.

```
let checkbox = document.getElementById('darkMode'); //prendo la mia checkbox
let state = false;
```

*//ad ogni click ottengo lo stato della checkbox e lo salvo nel local storage assieme alla data*

```
checkbox.addEventListener('click', function () {
  state = checkbox.checked;
  const obj = {
    darkMode: state,
    date: new Date().getTime() / (1000 * 60 * 60 * 24),
  }
  localStorage.setItem("mode", JSON.stringify(obj));
});
```

*//al caricamento della pagina recupero l'ultimo oggetto del LocalStorage*

```
document.addEventListener('DOMContentLoaded', function () {
  let retrievedObject = localStorage.getItem('mode');

  //pars ultimo oggetto, se darkMode = true cambio css
  let obj = JSON.parse(retrievedObject);

  let darkCSS = document.getElementById('darkCSS');
  let defCSS = document.getElementById('defCSS');
  if (obj != null) {
    if (obj.darkMode) {
      darkCSS.disabled = false;
      defCSS.disabled = true;
      checkbox.checked = true;
    } else {
      darkCSS.disabled = true;
      defCSS.disabled = false;
    }
  }
});
```

## AJAX

All'interno del file *getMeteo.js* ho implementato una chiamata *XMLHttpRequest* per ottenere il nome e le coordinate delle città suggerite.

Questi dati vengono letti da un file in formato JSON e ritornati tramite *request.responseText* il cui risultato è parsato per ottenere un oggetto JSON.

```
var request = new XMLHttpRequest();
request.open("GET", "../json/citta.json", false);
request.send(null);

if (request.readyState == 4 && request.status == 200) {
    var cittaJson = JSON.parse(request.responseText);
    [...]
```

Di seguito vengono riportate alcune righe del file *citta.json*:

```
[
  {
    "name": "Roma",
    "capital": true,
    "lat": "41.8933203",
    "lon": "12.4829321"
  },
  {
    "name": "Washington",
    "capital": true,
    "lat": "38.8950368",
    "lon": "-77.0365427"
  } [...]
```

## Api

Le chiamate API sono state utilizzate per ottenere le informazioni relative alle previsioni meteo tramite chiamata al servizio OpenWeather.

## Struttura della chiamata

`https://api.openweathermap.org/data/2.5/onecall?lat=" + cittaJson[i].lat + "&lon=" + cittaJson[i].lon + "&exclude=hourly,minutely&units=metric&appid=e21c453d380f0ca1bc5d071698438e15"`

La struttura prevede una prima parte che identifica il tipo di chiamata che si vuole effettuare *onecall versione 2.5*, latitudine e longitudine della città acquisite come mostrato sopra, in fine viene specificato l'esclusione di ore e minuti *exclude=hourly,minutely*, l'unità di misura *units=metric* e l'Id dell'api.

Per scopi di leggibilità al posto della struttura della chiamata verrà usato il placeholder "API".

## Risposta

```
var response = await fetch("API", {
    method: "GET"
});
```

Se la chiamata va a buon fine la variabile *response* conterrà una serie di informazioni.

```
var jsonObj = await response.json();
```

Da qui in avanti per ottenere le informazioni di *jsonObj* mi basterà usare la notazione puntata: *jsonObj.current.temp*

## Modalità

Le chiamate avvengono in modalità asincrona, questo si può notare dalle keyword *async* e *await*.

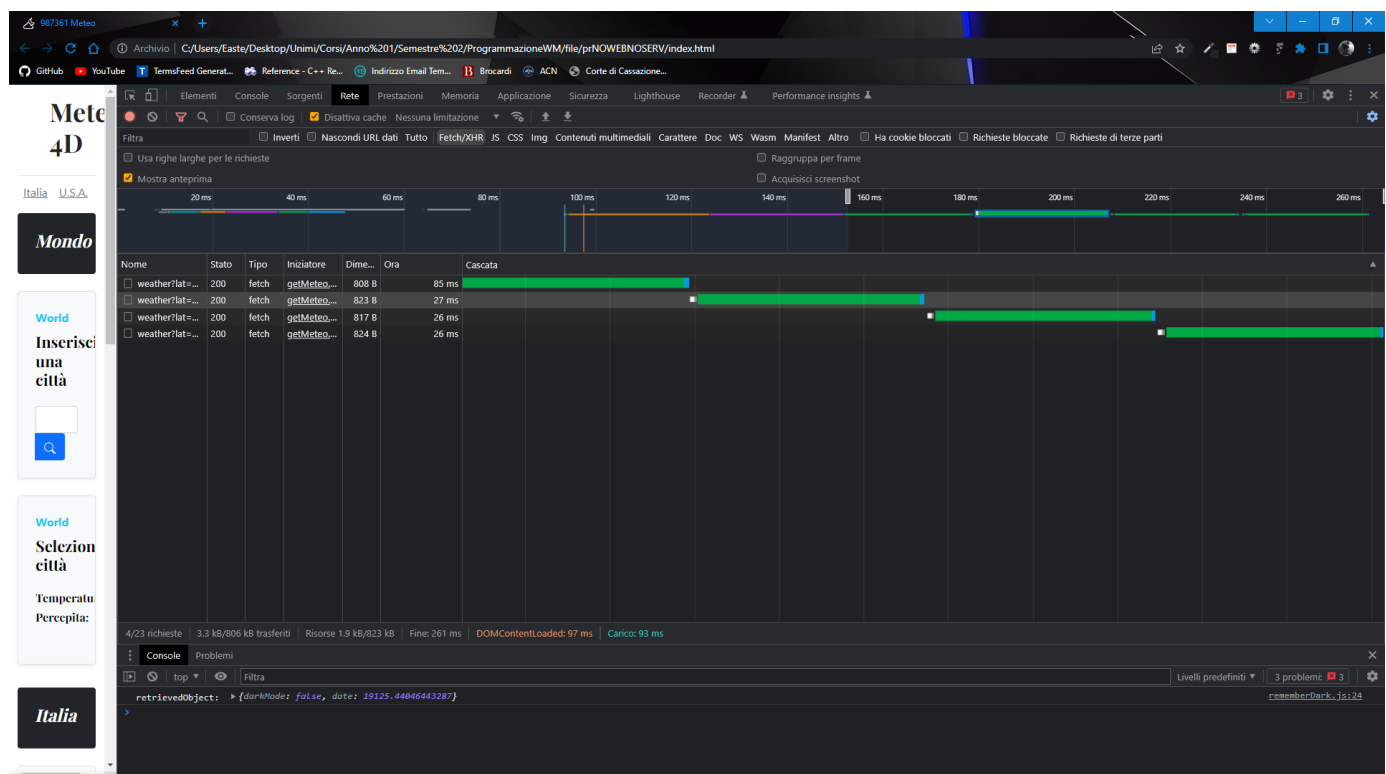
```
window.onload = async function getWeather() {
[...]
```

```
var response = await fetch([...])
```

## Parallelismo

In fase di sviluppo mi sono reso conto di un problema legato al parallelismo delle chiamate in quanto costituisce una grande fetta a livello di prestazioni.

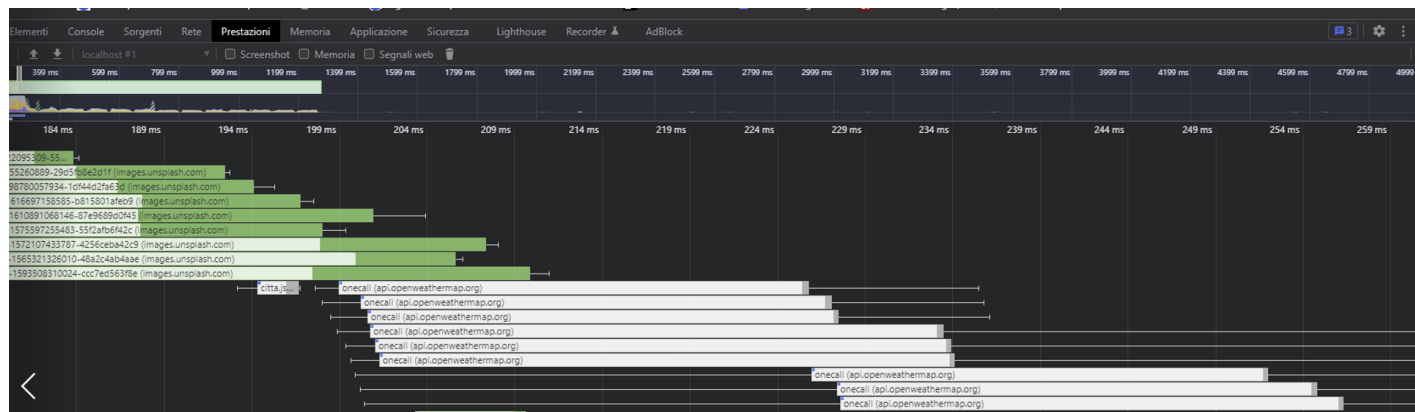
Nell'immagine sottostante vengono rappresentate le chiamate API in relazione al tempo, si può notare come vengano eseguite in maniera sequenziale.





Tramite l'introduzione della funzione *Promise.all* ho risolto questo problema, dal seguente frammento di codice si può inoltre notare come ogni richiesta sia identificata da un id.

```
const ids = [0, 1, 2, 3, 4, 5, 6, 7, 8]; // Array of ids
const responses = await Promise.all(
  ids.map(async i => {
    var response = await fetch("API", {
      method: "GET"
    });[...]
```



## Immagini

Per la gestione delle immagini mi sono avvalso di tue tecniche:

### Unsplash

Unsplash è un sito web che mette a disposizione un servizio API per ottenere un'immagine dato il nome della città `city.value`.

```
var response5 = await
fetch("https://api.unsplash.com/search/photos?client_id=JS6cTwzupVMwnF79Ii-aw7aZo-
vChivv5pYqEpgWBtQ&page=1&query=" + city.value + "&orientation=squarish", {
  method: "GET"
});
var jsonObj5 = await response5.json();
document.getElementById('imageSearchedPlace').setAttribute('src',
jsonObj5.results[0].urls.small);
```

Come nel caso di OpenWeather il risultato ottenuto è in formato JSON, procedo similamente ad ottenere l'url `jsonObj5.results[0].urls.small` e setto l'attributo html `src` con l'url appena ottenuto identificando la posizione tramite l'id.

### Web worker

Prendendo come riferimento la stessa immagine, nel primo caso senza e nel secondo con l'utilizzo del web worker possiamo notare una grande differenza in termini di caricamento della risorsa:

Richiesta di rete

URL [photo-1610891068146-87e9689d0f45](https://images.unsplash.com/photo-1610891068146-87e9689d0f45)

Durata 18.87 ms (13.23 ms trasferimento di rete + 5.64 ms caricamento risorsa)


Metodo di richiesta GET

Priorità Bassa

Tipo MIME image/avif

Dati codificati 182 kB

Corpo decodificato 181 kB



Dimensioni file: 181 kB

Origine corrente: <https://images.unsplash.com/photo-1610891068146-87e9689d0f45?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=735&q=80>

Anteprima

Richiesta di rete

URL [images.unsplash.com/photo-1610891068146-87e9689d0f45?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=735&q=80](https://images.unsplash.com/photo-1610891068146-87e9689d0f45?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=735&q=80)

Durata 17.27 ms (16.64 ms trasferimento di rete + 0.63 ms caricamento risorsa)

Metodo di richiesta GET

Priorità Alta

Tipo MIME image/jpeg

Dati codificati 203 kB

Corpo decodificato 203 kB

Iniziatore <blob:http://localhost:8080/6c0b2231-be5d-45fe-8642-ab9227622b97:6:30>

Questa differenza è dovuta al fatto che il web worker sfrutta un thread secondario che lavora in parallelo al principale.

webWorker.js

```
const workerAsText = document.querySelector('#worker').textContent
const workerAsBlob = new Blob([workerAsText], { type: 'text/javascript' })
const ImageLoaderWorker = new Worker(URL.createObjectURL(workerAsBlob))
const imgElements = document.querySelectorAll('img[data-src]')

ImageLoaderWorker.addEventListener('message', event => {
  // Grab the message data from the event
  const imageData = event.data
  const imageElement = document.querySelector(`img[data-src='${imageData.imageURL}']`)
  const objectURL = URL.createObjectURL(imageData.blob)
  imageElement.setAttribute('src', objectURL)
  imageElement.removeAttribute('data-src')
})

imgElements.forEach(imageElement => {
  const imageURL = imageElement.getAttribute('data-src')
  ImageLoaderWorker.postMessage(imageURL)
})
```

Index.html

```
<script id="worker" type="text/javascript-worker">
  self.addEventListener('message', async event => {
    const imageURL = event.data
    const response = await fetch(imageURL)
    const blob = await response.blob()

    // Send the image data to the UI thread!
    self.postMessage({
      imageURL: imageURL,
      blob: blob,
    })
  })
</script>
```

## Grafici

Per la realizzazione dei grafici è stato usato l'elemento html `<canvas>` il quale attraverso delle funzioni Javascript permette di realizzare un disegno.

### Index.html

```
<canvas class="my-4 w-100" id="myChart" width="900" height="300"></canvas>
```

Nel documento html specifico in ordine: le classi bootstrap che dovranno essere applicate, l'id che mi permetterà di collegare il canvas allo script, larghezza e altezza.

### GetMeteo.js

```
//controllo cambio del mese
```

```
function addDays(theDate, days) {  
    return new Date(theDate.getTime() + days * 24 * 60 * 60 * 1000);  
}  
  
var newDate = addDays(new Date(), 5);  
var ctx = document.getElementById('myChart' + (i + 2))  
var myChart = new Chart(ctx, {  
    type: 'line',  
    data: {  
        labels: [  
            date.getDate() + '/' + (addDays(date, 1).getMonth() + 1), //mese da 0 a 11  
            date.getDate() + 1 + '/' + (addDays(date, 2).getMonth() + 1),  
            date.getDate() + 2 + '/' + (addDays(date, 3).getMonth() + 1),  
            date.getDate() + 3 + '/' + (addDays(date, 4).getMonth() + 1),  
            date.getDate() + 4 + '/' + (addDays(date, 5).getMonth() + 1),  
            date.getDate() + 5 + '/' + (addDays(date, 6).getMonth() + 1),  
            date.getDate() + 6 + '/' + (addDays(date, 7).getMonth() + 1),  
        ],  
        datasets: [{  
            data: [  
                jsonObj.current.temp,  
                jsonObj.daily[1].temp.day,  
                jsonObj.daily[2].temp.day,  
                jsonObj.daily[3].temp.day,  
                jsonObj.daily[4].temp.day,  
                jsonObj.daily[5].temp.day,  
                jsonObj.daily[6].temp.day  
            ],  
            lineTension: 0,  
            backgroundColor: 'transparent',  
            borderColor: '#007bff',  
            borderWidth: 4,  
            pointBackgroundColor: '#007bff'  
        }]  
    },  
    [...]
```

Per prima cosa è stata implementata una funzione che permette di aggiungere dei giorni alla data corrente in modo da visualizzare sul grafico anche i giorni seguenti, tramite l'operazione `days * 24 * 60 * 60 * 1000` considero anche il cambio del mese in modo che il giorno seguente al 31/10 sia 1/11 e non 32/11.

Nella sezione `label` specifico la label che verrà visualizzata creando il formato gg/mm.

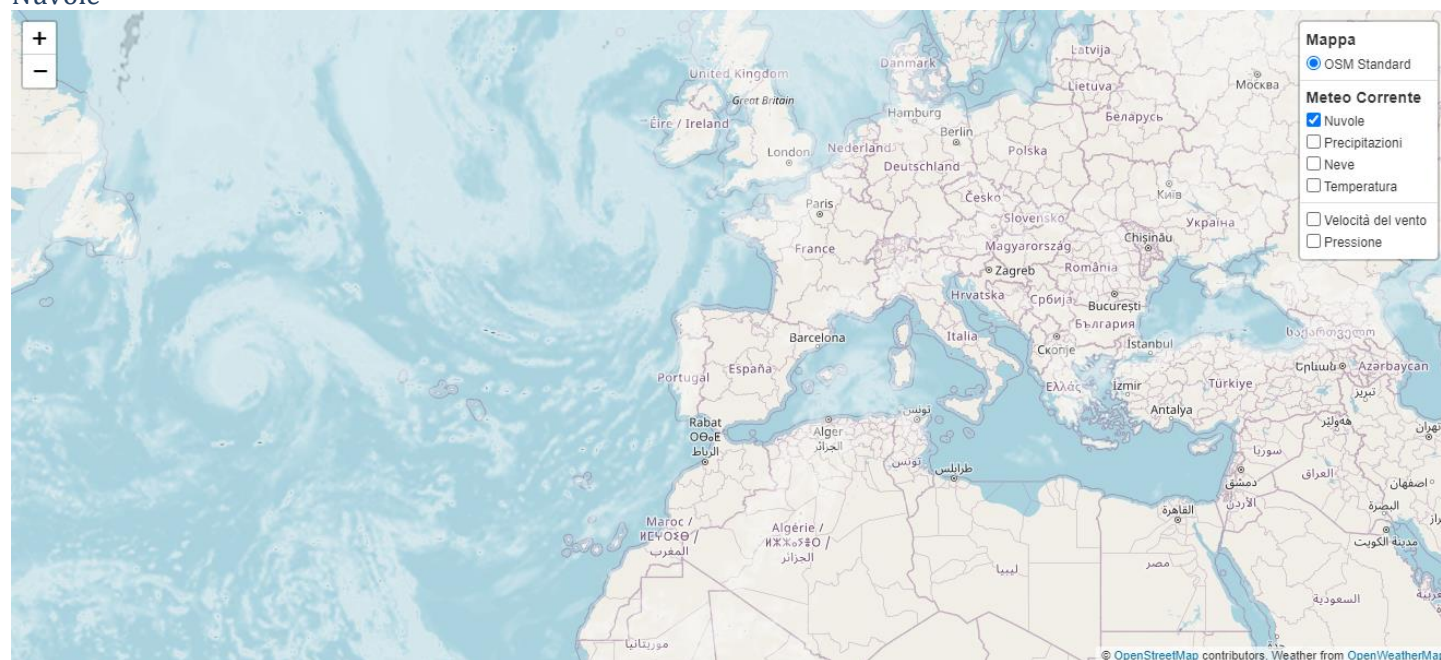
Nella sezione `data` visualizzo la temperatura ricavata da una API come visto in precedenza.

## Mappa

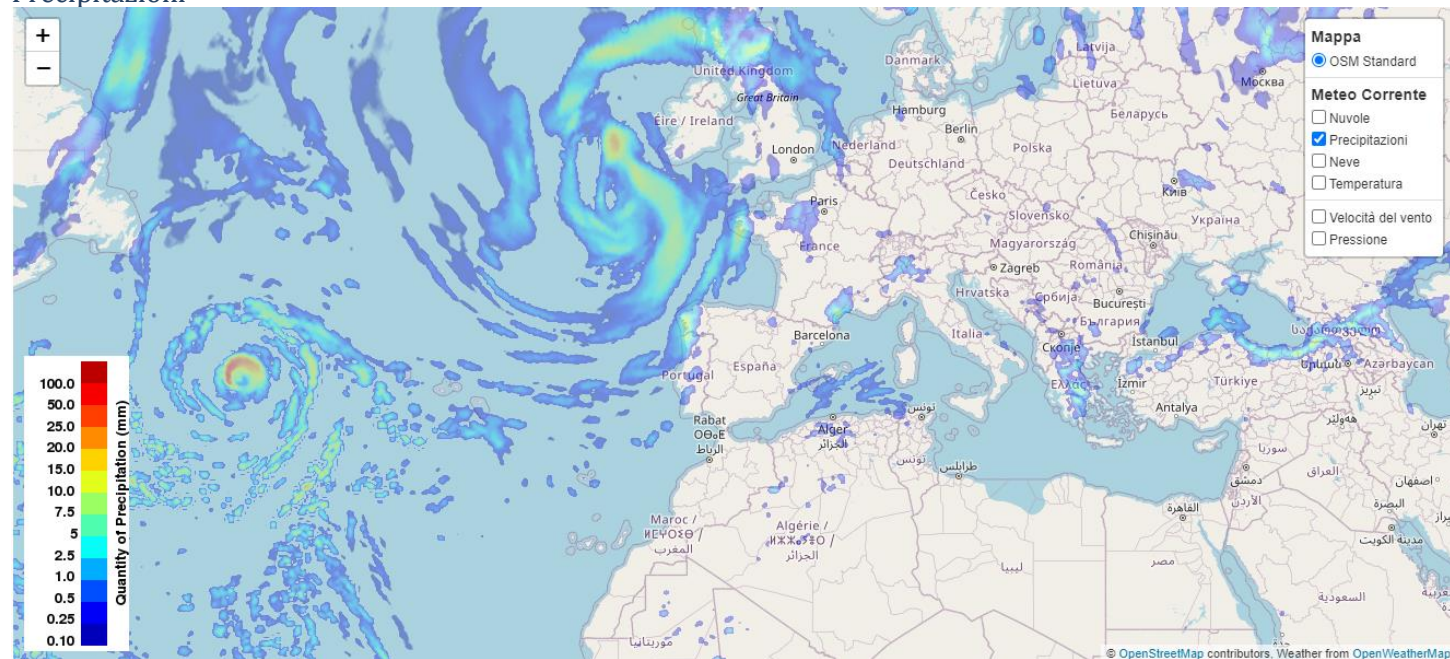
La mappa interattiva è stata scaricata del servizio OSM Open Street Map.

Tramite delle checkbox poste nella parte superiore-destra è possibile selezionate quale tipo visualizzazione avere.

### Nuvole

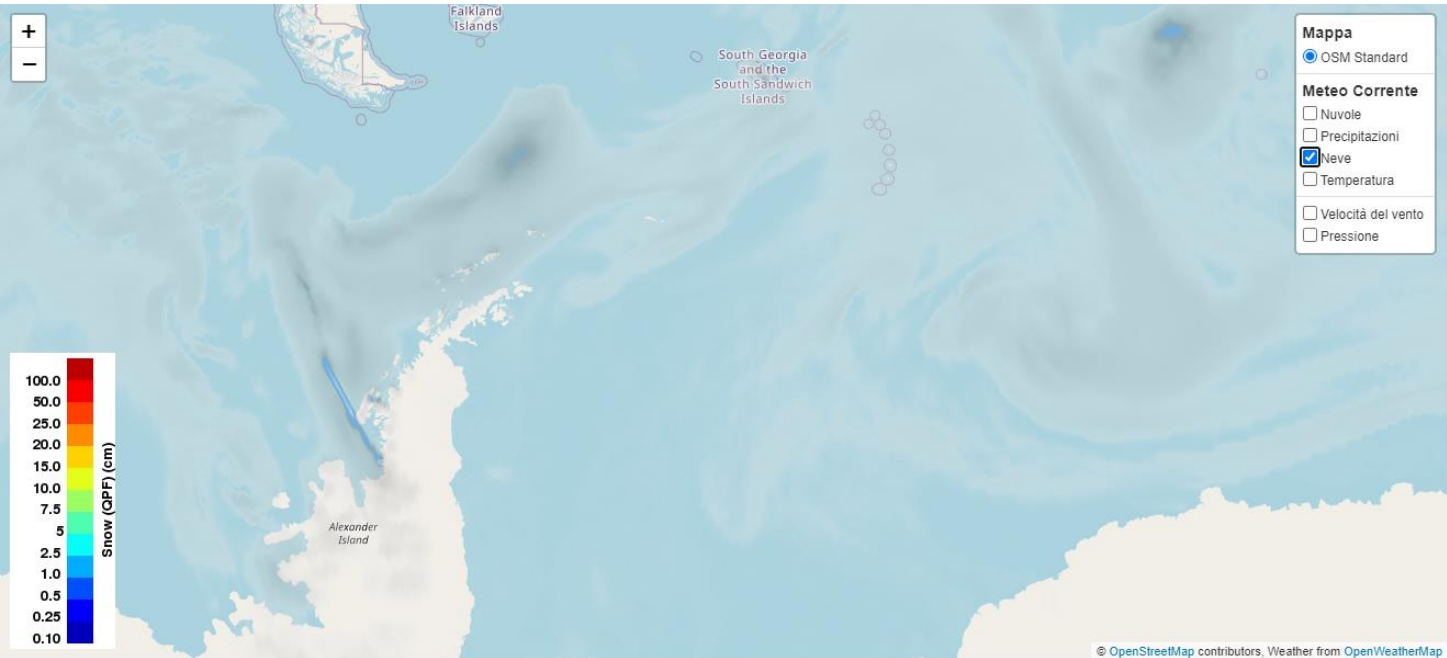


### Precipitazioni

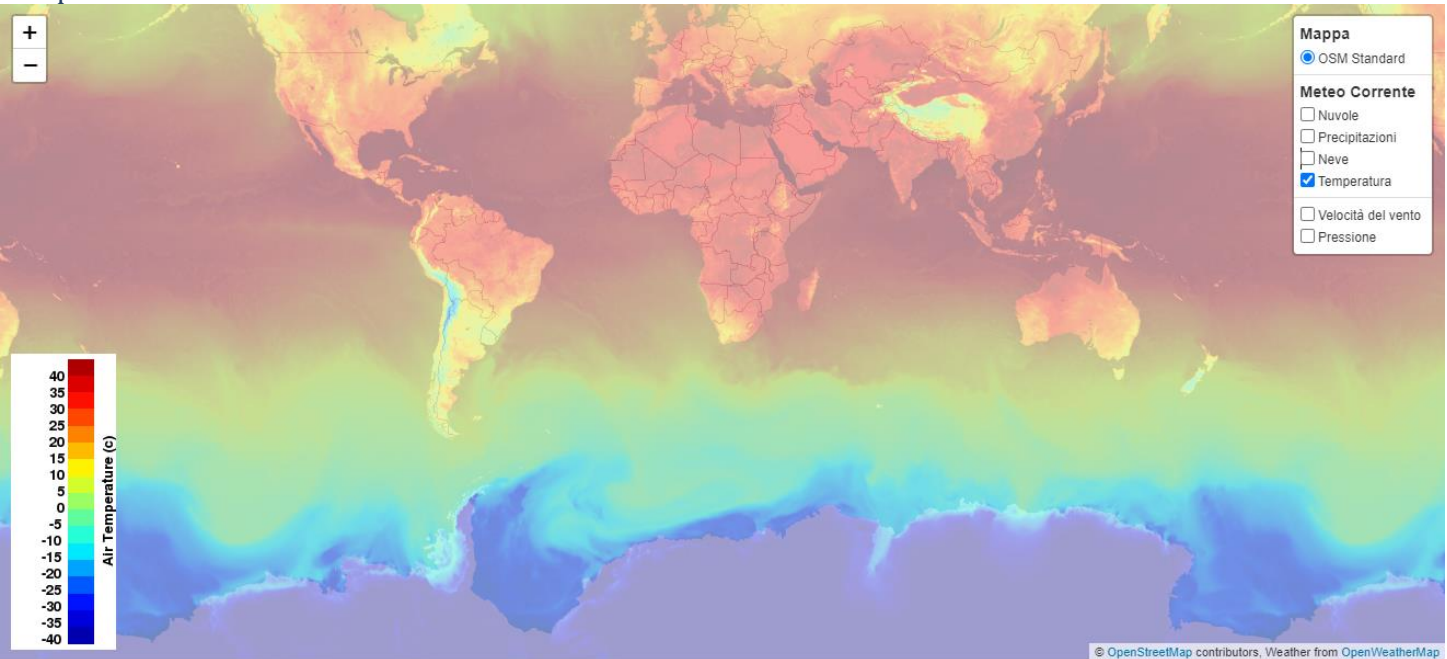




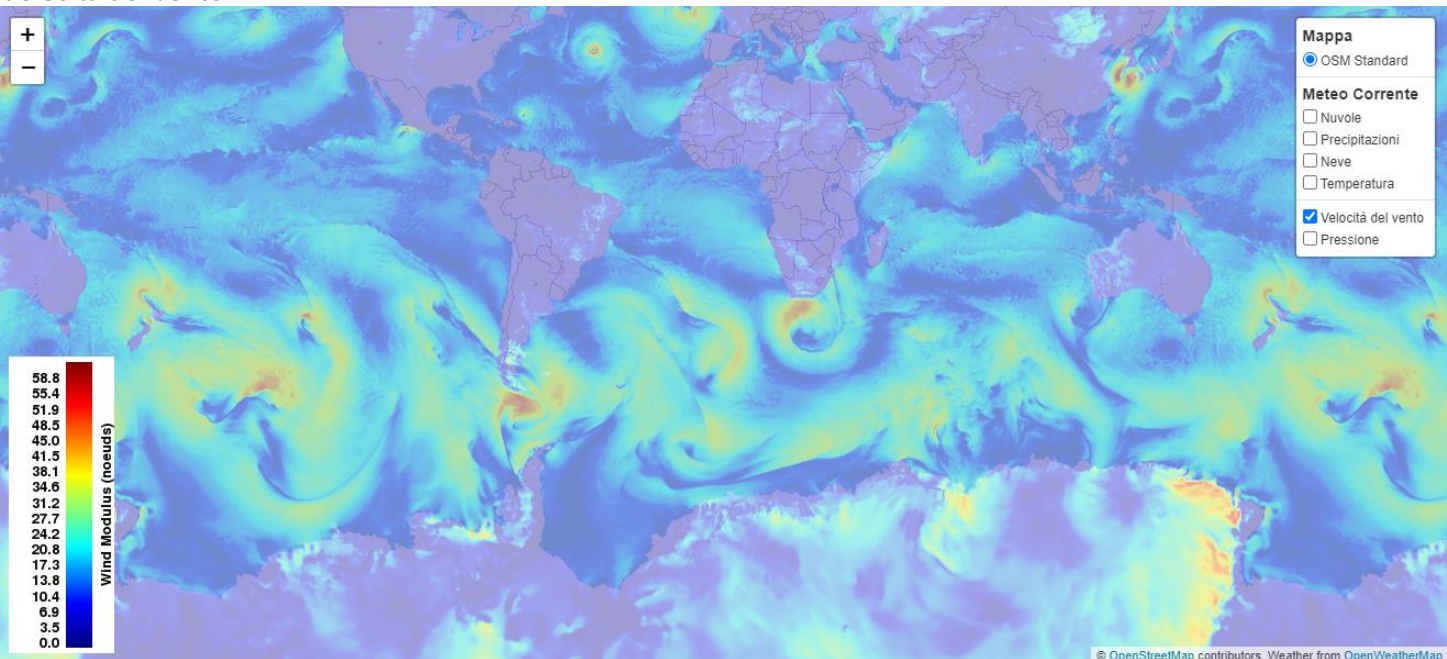
Neve



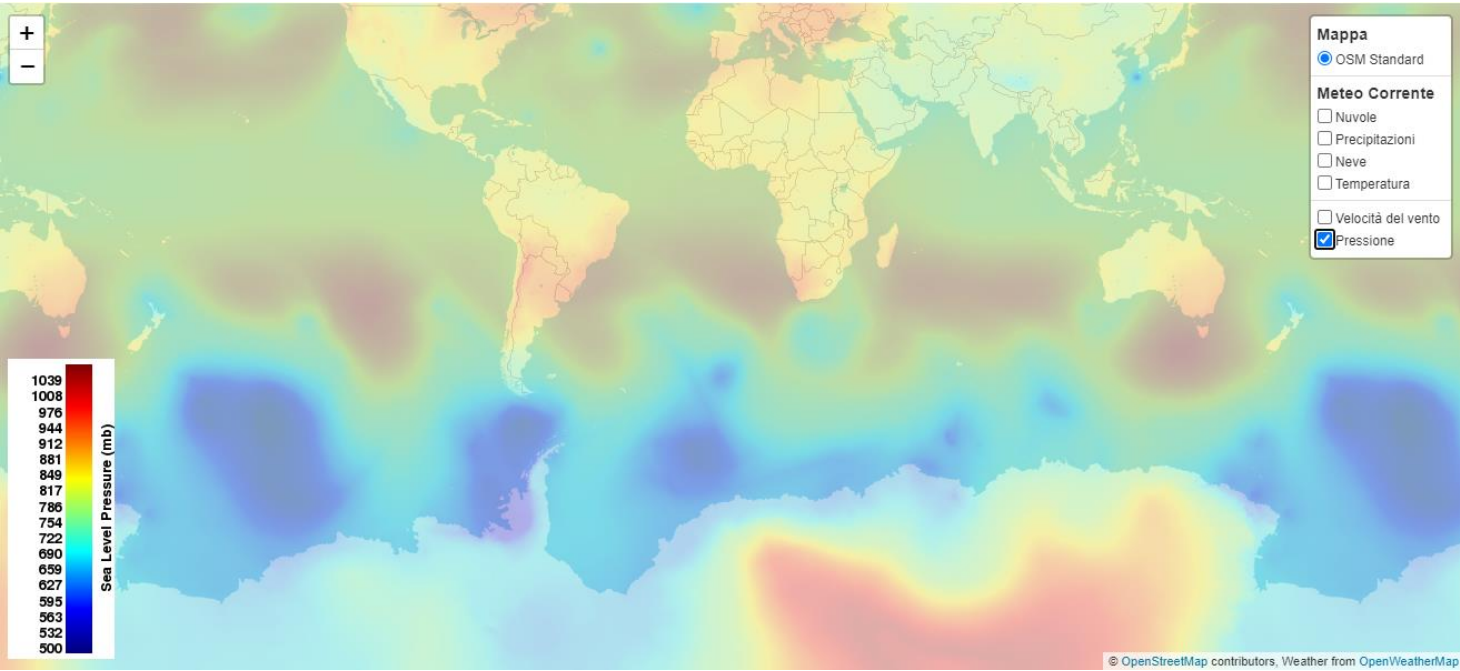
Temperatura



Velocità del vento



Pressione



## Sitografia

- <https://openweathermap.org/api>
- <https://unsplash.com/>
- <https://codepen.io/>
- <https://github.com/>
- <https://stackoverflow.com/>
- <https://ariel.unimi.it/>
- <https://schema.org>
- <https://www.lucarosati.it/blog/strategie-ricerca-informazione>
- <https://www.openstreetmap.org/user/Zartbitter>