

Meteo24-7 | Weather forecasts

Programmazione Web e Mobile

A.A. 2021-22

<http://meteo24-7.herokuapp.com/>



Indice

1. Introduzione
 - 1.1 Breve analisi dei requisiti
 - 1.1.1 Destinatari
 - 1.1.2 Modello di valore
 - 1.1.3 Flusso dei dati
 - 1.1.4 Aspetti tecnologici
 2. Interfacce
 - 2.1 Panoramica interfaccia utente
 3. Architettura
 - 3.1 Diagrammi
 - 3.1.1 Funzionamento index.ejs
 - 3.1.2 Funzionamento meteo.ejs
 4. Codice
 - 4.1 HTML
 - 4.2 CSS
 - 4.3 JavaScript
 - 4.4 app.js
 - 4.5 routes.js
 5. Prestazioni
 - 5.1 Miglioramenti
 6. Usabilità
 7. Deploy sul web
 8. Conclusioni
 9. Sitografia

1. Introduzione

Meteo24-7 è un'applicazione web che consente di ottenere informazioni riguardanti il meteo di qualsiasi città del mondo.

È costituita da una pagina iniziale che funge da *landing page*, la quale contiene un link per la relativa repository di GitHub(che contiene il codice) e un bottone con il collegamento alla pagina contenente le informazioni meteorologiche (*templating* con ejs) come: tempo, umidità, velocità del vento, alba, tramonto, gradi percepiti.

In aggiunta, presenta una preview dei prossimi sei giorni con relativa temperatura minima e massima e un'immagine che indica le condizioni climatiche (informazioni e immagini ottenute con *OpenWeather API*).

Infine è stato fatto il deploy sul web tramite *herokuapp* per far sì che la sua ricerca sia possibile da qualsiasi dispositivo.

1.1 Breve analisi dei requisiti

1.1.1 Destinatari

Meteo24-7 è destinata ad essere utilizzata da quegli utenti che necessitano informazioni meteorologiche di una città per vari motivi: per curiosità riguardo le informazioni correnti della propria città, per futuri viaggi o per interesse personale.

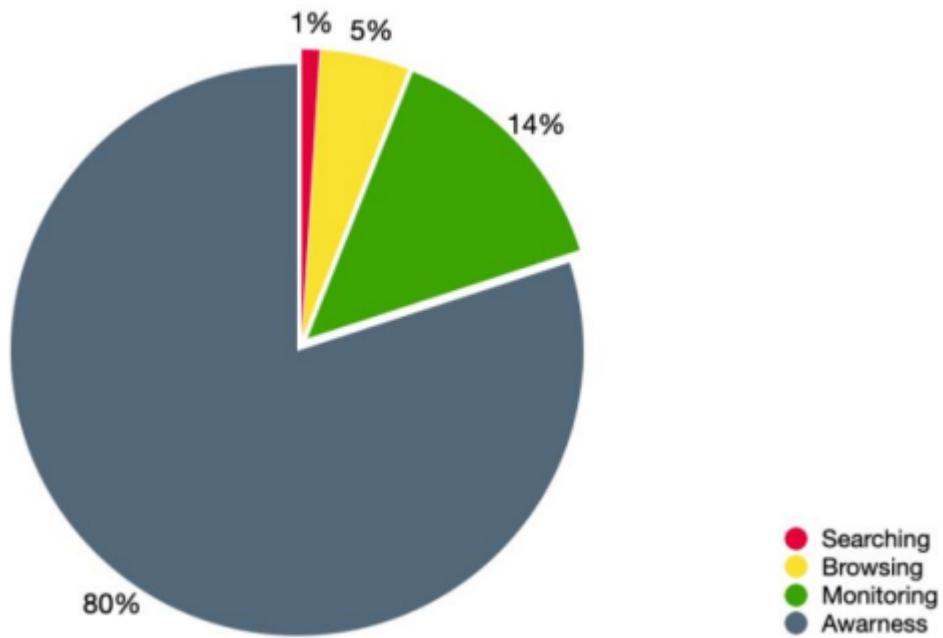
L'ipotetico destinatario non necessita di particolari conoscenze informatiche o scientifiche per l'utilizzo del servizio, conosce già cosa vuole cercare e ha un livello di motivazione medio, caratterizzato da curiosità e necessità.

La landing page è utilizzata come introduzione all'applicazione web e come "piattaforma di lancio" verso la pagina contenente le informazioni del meteo. Il contenuto è presentato in inglese così da poter raggiungere un bacino di utenza maggiore e internazionale, nonostante le informazioni generali sul meteo della località siano comprensibili a tutti perché sotto forma di immagine.

Il design è responsive, questo vuol dire che il contenuto può essere acceduto e visualizzato da qualsiasi tipo di dispositivo (monitor, tablet, smartphone...).

Per capire meglio come gestire la presenza di più tipologie di destinatari occorre usare una classificazione come il *modello di Marcia Bates*, che divide le strategie di ricerca dell'informazione in quattro tipologie principali:

- **Directed–Active:** searching, quando si ricerca una città per conoscere le informazioni riguardanti il clima per un eventuale viaggio programmato nei giorni successivi.
- **Directed–Passive:** monitoring, quando si apre la pagina meteo per controllare periodicamente la temperatura della città in cui ci si trova.
- **Undirected–Active:** browsing, tramite l'indicizzazione delle pagine dai motori di ricerca, nel caso in cui venga cercato un servizio meteo; Meteo24-7 potrebbe apparire come risultato della ricerca.
- **Undirected–Passive:** awareness, tramite un'acquisizione passiva e indiretta delle informazioni, ad esempio tramite pubblicità, passaparola e/o condivisione.



1.1.2 Modello di valore

Meteo24-7 non è a scopo di lucro, è stato ideato per essere gratuito e senza la presenza di invasive pubblicità e pop-up.

Allo stesso tempo è un chiaro esempio di come per inserirsi nel mondo del web e nell'economia dell'informazione non occorrono ingenti somme di denaro o finanziamenti.

Tuttavia un modo per ottenere guadagno da questa applicazione potrebbe essere l'inserimento di pubblicità o ads che spostano il valore di rendimento dal volume di traffico e di utenza visitante generato.

1.1.3 Flusso dei dati

I dati sono elementi indipendenti dalla struttura della pagina e hanno quindi un ciclo di vita indipendente e separato.

Il contenuto deve essere percepito come d'interesse per i destinatari ed è stato quindi espresso in uno stile adeguato, garantendo comunque il massimo dell'informazione ricercata.

Il punto di scambio dati tra browser (client) e server avviene nel momento della ricerca della città, dove viene inviato al server, con metodo POST, il luogo cercato. Viene recuperato dal server attraverso una funzione di *express* che permette di accettare i dati della POST accedendo al suo body (`app.use(express.urlencoded({ extended: true }))`). I dati vengono recuperati dal servizio OpenWeather che mette a disposizione delle API. Le call API vengono effettuate nel file `routes.js`. Per lo sfondo, invece, viene utilizzata una call API al servizio di Unsplash, il quale restituisce una serie di immagini relative alla query effettuata.

1.1.4 Aspetti tecnologici

- **HTML5**

Utilizzando HTML5 sono state create le strutture delle due pagine presenti, con gli elementi principali all'interno di esse.

Esse sono state validate tramite “[The W3C Markup Validation Service](#)”

Al loro interno sono stati importati diversi file esterni, (CSS per icone e per font), e interni che descrivono lo stile delle pagine e permettono di creare la DarkMode e le animazioni.

La landing page manda tramite il bottone alla pagina del meteo tramite collegamento ipertestuale.

- **CSS3**

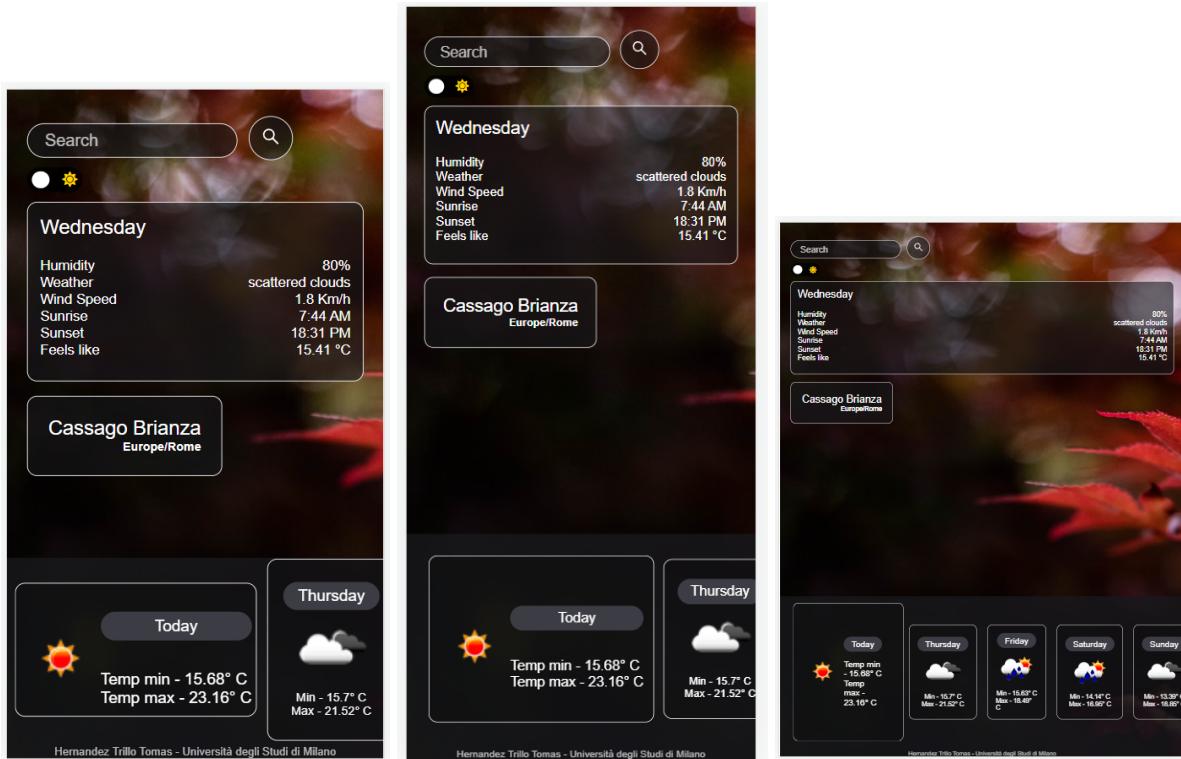
Utilizzando CSS3 è stato possibile definire precisamente le strutture presenti nelle pagine, “delineate” solamente nei file HTML5 e prendendo finalmente forma tramite gli attributi specificati nei fogli di stile.

Con CSS3 è stato creato uno stile minimale, semplice ma molto intuitivo ed è stata gestita la responsiveness dell'applicazione web, adattandola alle dimensioni di qualsiasi dispositivo tramite le *media query*.

Inoltre è stato utile per la creazione delle animazioni presenti in entrambe le pagine che le rendono interattive e dinamiche.

Sfruttando strumentazioni utili come “UnCSS”, sono stati rimossi segmenti di codice inutilizzati e quindi inutili.

Responsive design (iPhoneSE, Pixel 5 e Ipad Mini):



● EJS

Il progetto sfrutta un semplice linguaggio di templating chiamato EJS (Embedded Javascript templates).

Questo linguaggio ha permesso di creare una pagina dinamica che assuma aspetti diversi in base ai dati ricevuti. Era indispensabile utilizzare questa tecnologia in quanto bisogna mostrare dati diversi in base alla città cercata.

Viene integrata direttamente all'interno del codice HTML con tag come `<%= %>` o `<% %>`.

I dati da renderizzare vengono specificati all'interno di ‘routes.js’.

● JavaScript

Le funzioni principali ed il back end dell'applicazione sono state rese possibili grazie a JS, che è stato utilizzato sia lato Client che lato Server.

Lato Client ad esempio per la selezione della DarkMode o della LightMode, e lato Server con l'utilizzo di Node.js e per le chiamate API, per poter usufruire dei dati in tempo reale del meteo di tutto il globo e per conoscere l'IP dell'utente che visita l'applicazione web.

● Node

Sono stati utilizzati diversi moduli di Node.js, come:

- 1) **Express:** utile per la gestione del routing, delle views e il deploying del server
All'interno di 'routes.js' è presente la logica di routing per la realizzazione della renderizzazione della view 'Meteo.ejs', con i dati richiesti aggiornati dinamicamente al suo interno.
In 'app.js' sono presenti le funzioni di *middleware* ed il deploy sul server.
- 2) **Node-fetch:** è un modulo che permette l'uso di "fetch" lato server.
Fetch permette di effettuare le chiamate alle API utilizzate, ricevendo i dati come risposta; è considerato l'evoluzione di XMLHttpRequest perché rende più semplice effettuare richieste *asynchronous* e gestire meglio le response.
Cosa significa che le richieste sono asincrone? significa che le variabili che contengono la risposta vengono inizializzate solamente quando la risposta è pronta, nel frattempo conterranno una "Promise". Tutto questo va indicato tramite le keywords 'async' e 'await', così che l'esecuzione di quella funzione aspetti la risposta con i dati prima di poter proseguire.
- 3) **Dotenv:** si occupa di caricare tutte le variabili di ambiente contenute nel file '.env' dentro 'process.env'.
Le informazioni contenute nel file '.env' sono le API key e la porta del server in caso non ce ne siano altre disponibili.
Questo fornisce anche una maggiore segretezza di queste ultime.

● API

Le API che sono state utilizzate sono:

- 1) **OneCall API 1.0 di Open Weather**, che restituisce, in formato JSON, tutti i dati relativi al meteo del giorno corrente e dei 6 giorni successivi.
I dati utilizzati sono l'umidità, una descrizione del cielo, la velocità del vento, l'orario dell'alba e del tramonto, la temperatura percepita, la temperatura massima, quella minima e l'icona del clima.
- 2) **Geocoding API di OpenWeather**, che permette, a partire dal nome di una città, di ricevere latitudine e longitudine corrispondenti.
Queste sono necessarie in quanto OneCallAPI 1.0 necessita del loro utilizzo.
- 3) **Unsplash API**, che restituisce un elenco di immagini relative alla parola ricercata.
Usata per ottenere un background diverso della città corrispondente all'IP dell'utente, (per avere uno sfondo, non casuale, differente ad ogni accesso), e uno sfondo corrispondente alla città cercata ogni volta che si effettua una ricerca
- 4) **IP-API**, che restituisce informazioni relative ad un indirizzo IP.
In questa applicazione web è stata usata per recuperare le coordinate dell'IP del client mediante una richiesta GET a 'meteo.ejs', per poi ottenere e mostrare i dati relativi alla località corrispondente.

● Herokuapp

Herokuapp è l'architettura che ha permesso il deploy dell'applicazione web su internet, fornendo un URI raggiungibile da qualsiasi dispositivo.

È un'architettura sviluppata da *Salesforce* che permette di fare il deploy sul web di una repository di GitHub e fornisce, oltre ad altre tecnologie utili, un sistema di log in tempo reale, utile per il debugging.

L'URI diretto al sito web è meteo24-7.herokuapp.com

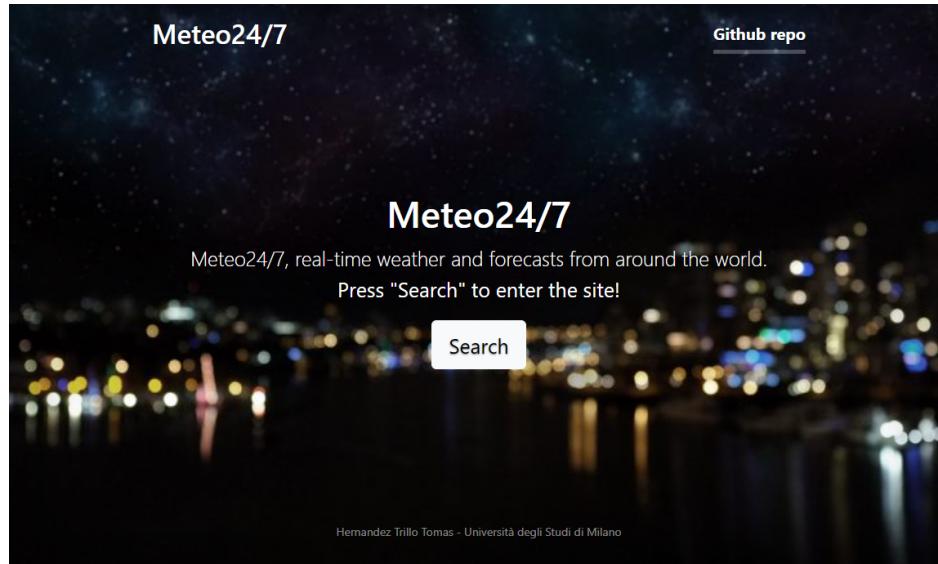
2. Interfacce

Meteo24-7 ha due interfacce, che sono state prima progettate a livello grafico per avere il miglior risultato a livello comunicativo, e successivamente realizzate mediante HTML5 e CSS3:

- 1) **Index:** è una landing page che offre una presentazione del sito, un link alla repository di GitHub che contiene il codice sorgente, un bottone per accedere al servizio meteo e un footer con le informazioni principali riguardo il suo sviluppatore.
Inoltre ha delle animazioni per attirare l'attenzione dell'utente e rendere l'esperienza di visita dell'applicazione più dinamica.
- 2) **Meteo:** è la pagina che contiene il servizio vero e proprio, dando la possibilità all'utente di cercare la città di cui desidera ricevere informazioni.
Presenta una barra di ricerca con un campo impostato a *required*, quindi obbligatorio da compilare, uno switch per la *darkmode*, una sezione adiacente contenente informazioni su umidità, tempo, velocità del vento, alba, tramonto e gradi percepiti.
In basso una sezione contenente un'immagine e la temperatura minima e massima del giorno corrente e dei successivi sei giorni.
Sono presenti delle animazioni allo spostamento del cursore (:hover) su alcune sezioni per rendere la visita dell'applicazione interattiva e dinamica.
Al caricamento della pagina vengono mostrate all'utente informazioni riguardanti la città rilevata dall'acquisizione dell'indirizzo IP e quindi dalla relativa longitudine e latitudine per poter eseguire una chiamata API e impaginare le informazioni.
Alla ricerca di una città da parte dell'utente le informazioni vengono sostituite da quelle corrette grazie al templating con ejs.

2.1 Panoramica interfaccia utente

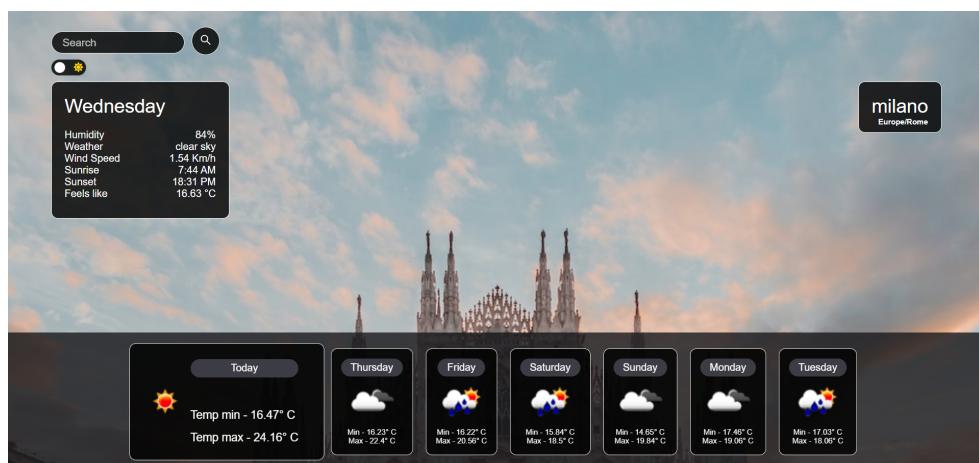
index.ejs



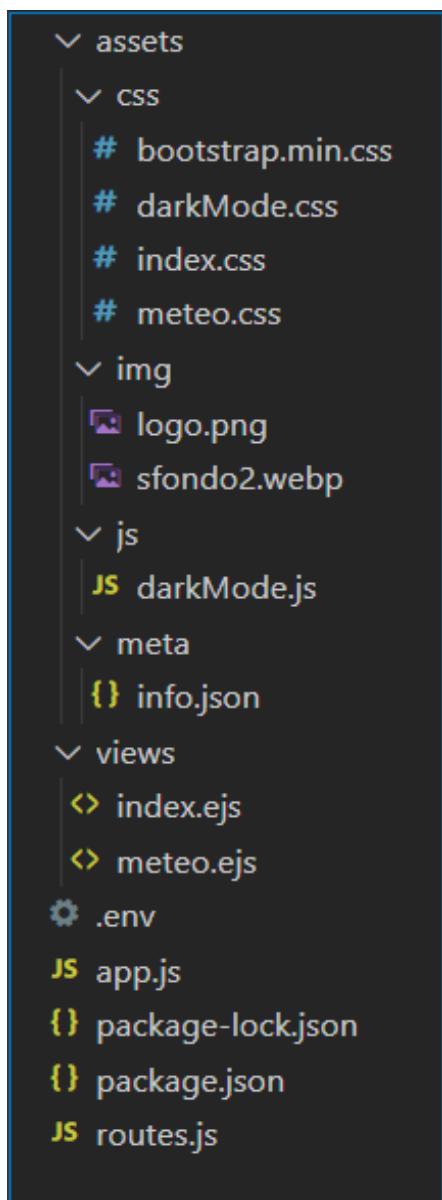
meteo.ejs (LightMode)



meteo.ejs (NightMode)



3. Architettura



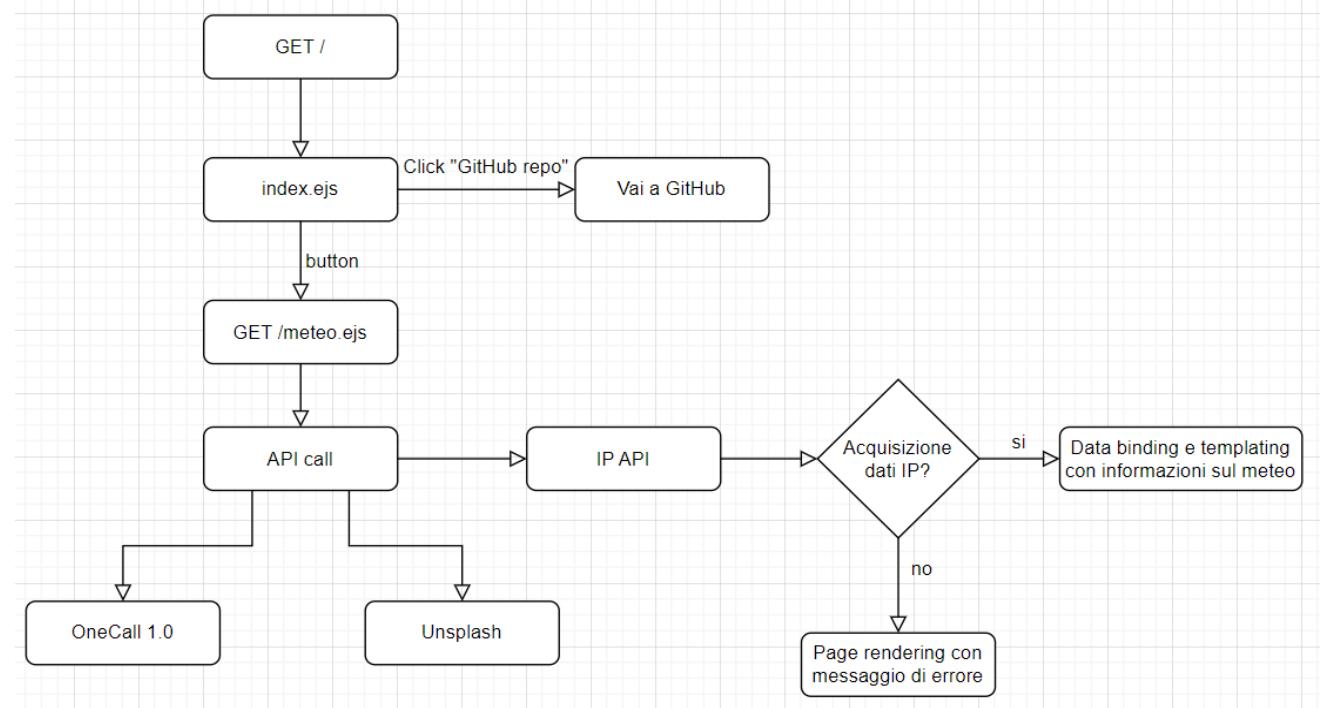
Il progetto è composto dalla cartella ‘assets’ che contiene i fogli di stile,(bootstrap.min è stato utilizzato per creare lo stile di index.ejs scegliendo il template ‘cover’), le immagini utilizzate, i file JavaScript e uno schema json che contiene le informazioni per l’indicizzazione della applicazione web sui motori di ricerca, creato con lo schema WebPage presente su [schema.org](#).

In seguito c’è *views* che contiene le interfacce utente, quindi i file .ejs dove avviene il data binding con le informazioni inerenti alla città cercata. In ‘.env’ sono contenute le variabili d’ambiente (API key e numero di porta del server). ‘package-lock.json’ e ‘package.json’ contengono tutti i moduli necessari per il funzionamento di node.js

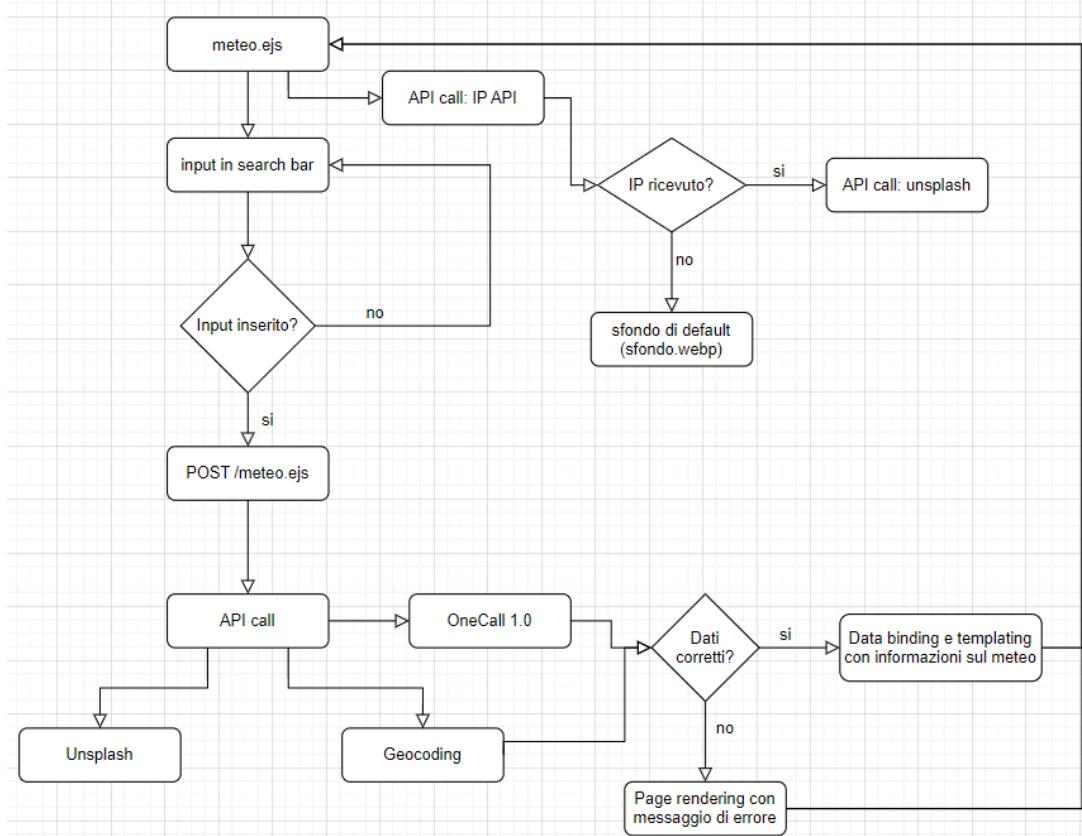
Infine in ‘app.js’ si può trovare la configurazione di un server di tipo express e le funzioni di middleware mentre ‘routes.js’ contiene la gestione del routing e quindi degli endpoint raggiungibili dall’utente, con il rendering delle views e le chiamate API importate in seguito con le chiamate ejs.

3.1 Diagrammi

3.1.1 Funzionamento index.ejs



3.1.2 Funzionamento meteo.ejs



4. Codice

4.1 HTML

index.ejs: composto da navbar, descrizione e pulsante e footer

```
<body class="d-flex h-100 text-center text-white bg-dark">
  <div class="cover-container d-flex w-100 h-100 p-3 mx-auto flex-column">
    <!-- navbar -->
    <header class="mb-4">
      <div>
        <h3 class="float-md-start mb-0">Meteo24/7</h3>
        <nav class="nav nav-masthead justify-content-center float-md-end">
          <a class="nav-link fw-bold py-1 px-0 active" href="https://github.com/TomasHernandez1/Meteo24-7" target="_blank">Github repo</a>
        </nav>
      </div>
    </header>
    <!-- descrizione e pulsante -->
    <main class="px-3">
      <h1 class="animate-left typewriter">Meteo24/7</h1>
      <p class="animate-left lead">Meteo24/7, real-time weather and forecasts from around the world.<br><b>Press "Search" to enter the site!</b>
      <p class="lead">
        <a href="/meteo" class="btn btn-lg btn-light btn-info">Search</a>
      </p>
    </main>
    <!-- footer -->
    <footer class="mt-auto text-white-50" id="foot">
      <p>Hernandez Trillo Tomas - Università degli Studi di Milano</p>
    </footer>
  </div>
</body>
```

meteo.ejs: composto da barra di ricerca, toggle darkmode, info principali e meteo di oggi e dei giorni successivi

```
<!-- info giorno -->
<div class="current-info">
  <div class="date-container">
    <div class="others" id="current-weather-items">
      <!-- info oggi -->
      <div class="date" id="date"><%= day1 %></div><br>
      <div class="weather-item">
        <div>Humidity</div>
        <div><%= humidity %>%</div>
      </div>
      <div class="weather-item">
        <div>Weather</div>
        <div><%= description %></div>
      </div>
      <div class="weather-item">
        <div>Wind Speed</div>
        <div><%= wind %> Km/h</div>
      </div>
      <div class="weather-item">
        <div>Sunrise</div>
        <div><%= sunrise %> AM</div>
      </div>
      <div class="weather-item">
        <div>Sunset</div>
        <div><%= sunset %> PM</div>
      </div>
      <div class="weather-item">
        <div>Feels like</div>
        <div><%= feels %> °C</div>
      </div>
    </div>
  </div>
  <!-- città e fuso-orario-->
  <div class="place-container" id="place-container">
    <div class="time-zone" id="time-zone"><%= city %></div>
    <div id="country" class="country"><%= timezone %></div>
  </div>
```

Qua possiamo osservare le chiamate .ejs per la realizzazione del templating.

4.2 CSS

Esempio *media query*

```
@media only screen and (max-width: 400px){
    .future-forecast{
        justify-content: start;
        align-items: none;
        overflow-x: scroll;
        overflow-y: clip
    }

    #current-weather-item{color: black !important}

    ::-webkit-scrollbar{
        width:0px;
        background: transparent
    }

    .future-forecast .today .temp{
        padding-top:0;
        width: 100%
    }
    .future-forecast .today{
        width:75%;
        height:100%;
        padding-right:0px;
        padding-left:0px;
        padding-top:70px;
        padding-bottom:50px
    }
    .weather-forecast{width:26%}
}
```

4.3 JavaScript

Chiamata della darkMode

```
document.addEventListener('DOMContentLoaded', function () {
    var checkbox = document.getElementById('switch');

    checkbox.addEventListener('change', function () {
        if(checkbox.checked) {
            setLight()
        } else {
            setDark()
        }
    });
});
```

4.4 app.js

Deploy di un server express, funzioni di *middleware* e import delle routes

```
const express = require('express')
const app = express()
const Ddos = require('ddos')
var ddos = new Ddos ({burst:10, limit:10})

//Import routes
const routes = require('./routes.js')

app.use(express.urlencoded({ extended: true }))
app.use(ddos.express)

//Use view engine
app.set('view engine', 'ejs')

//Middleware route
app.use('/', routes)
app.use(express.static('assets'))

const PORT = process.env.PORT || 5000

app.listen(PORT, () => {
  console.log(`Server starting at ${PORT}`)
})
```

4.5 routes.js

Chiamate POST /meteo.ejs con rendering della pagina

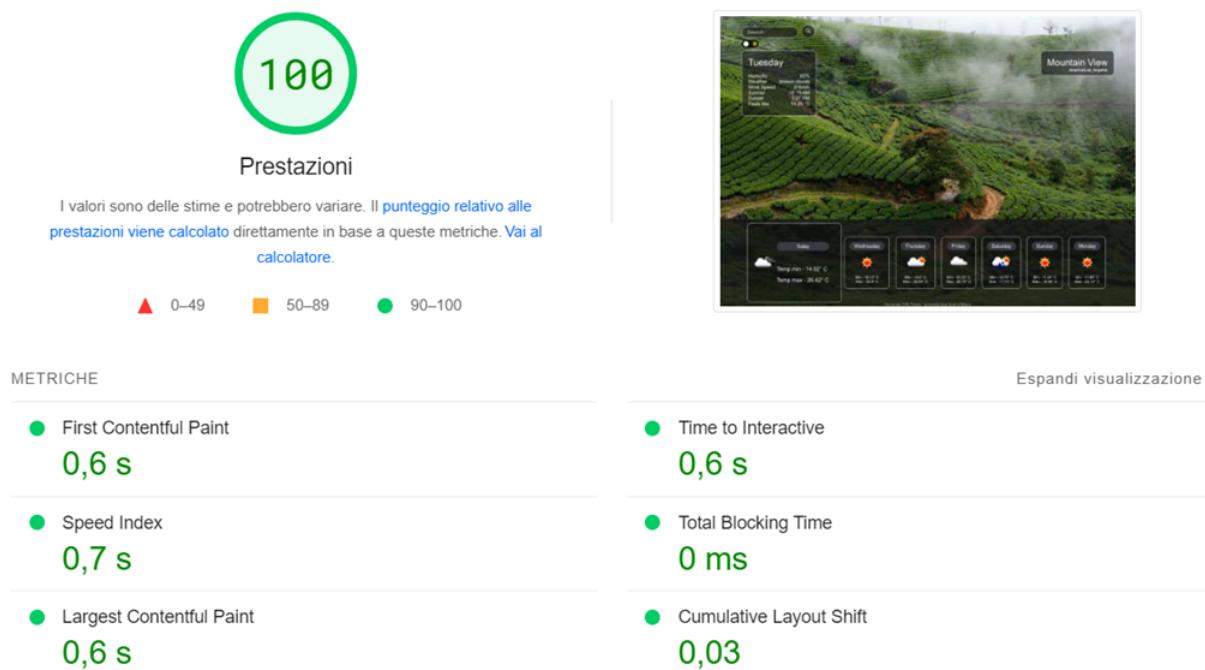
```
res.render('meteo', {
  city: city,
  temp: data.current.temp,
  timezone: data.timezone,
  description: data.current.weather[0].description,
  humidity: data.current.humidity,
  wind: data.current.wind_speed,
  imgsrc: "https://openweathermap.org/img/w/" + data.current.weather[0].icon + ".png",
  aq: index[aq-1],
  min: data.daily[0].temp.min,
  max: data.daily[0].temp.max,
  mintom: data.daily[1].temp.min,
  maxtom: data.daily[1].temp.max,
  mindayafter: data.daily[2].temp.min,
  maxdayafter: data.daily[2].temp.max,
  min3: data.daily[3].temp.min,
  max3: data.daily[3].temp.max,
  min4: data.daily[4].temp.min,
  max4: data.daily[4].temp.max,
  min5: data.daily[5].temp.min,
  max5: data.daily[5].temp.max,
  min6: data.daily[6].temp.min,
  max6: data.daily[6].temp.max,
  imgtoday: "https://openweathermap.org/img/w/" + data.daily[0].weather[0].icon + ".png",
  imgtom: "https://openweathermap.org/img/w/" + data.daily[1].weather[0].icon + ".png",
  imgdayafter: "https://openweathermap.org/img/w/" + data.daily[2].weather[0].icon + ".png",
  imgday3: "https://openweathermap.org/img/w/" + data.daily[3].weather[0].icon + ".png",
  imgday4: "https://openweathermap.org/img/w/" + data.daily[4].weather[0].icon + ".png",
  imgday5: "https://openweathermap.org/img/w/" + data.daily[5].weather[0].icon + ".png",
  imgday6: "https://openweathermap.org/img/w/" + data.daily[6].weather[0].icon + ".png",
  unsplash: backgroundLink,
  day1: days[(date.getDay()) % 7],
  day2: days[(date.getDay() + 1) % 7],
  day3: days[(date.getDay() + 2) % 7],
  day4: days[(date.getDay() + 3) % 7],
  day5: days[(date.getDay() + 4) % 7],
  day6: days[(date.getDay() + 5) % 7],
```

5 Prestazioni

Il caricamento della pagina index.ejs risulta molto veloce, essendo principalmente statica. E' stato migliorato passando da uno sfondo .jpg (più pesante e lento) ad uno sfondo .webp (formato più leggero e versatile)

Per meteo.ejs inizialmente, per impostare un background casuale per ogni richiesta, veniva usato source.unsplash, un'alternativa alla normale API call. Visti i lunghi tempi di caricamento (circa 3s), si è passati all'utilizzo della comune call API al servizio di unsplash, notando un forte guadagno di prestazione nel caricamento dello sfondo della pagina meteo.ejs e riducendo anche il layout shift (movimenti non causati dall'utente)

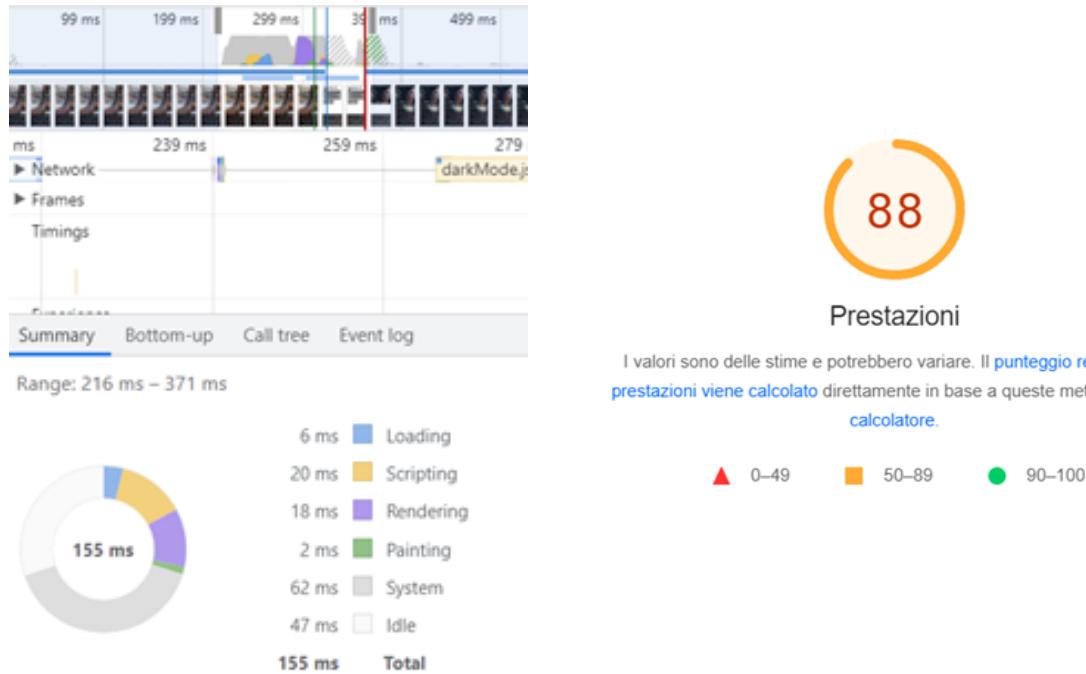
Risultati dell'analisi "LightHouse Report by Google":



5.1 Miglioramenti

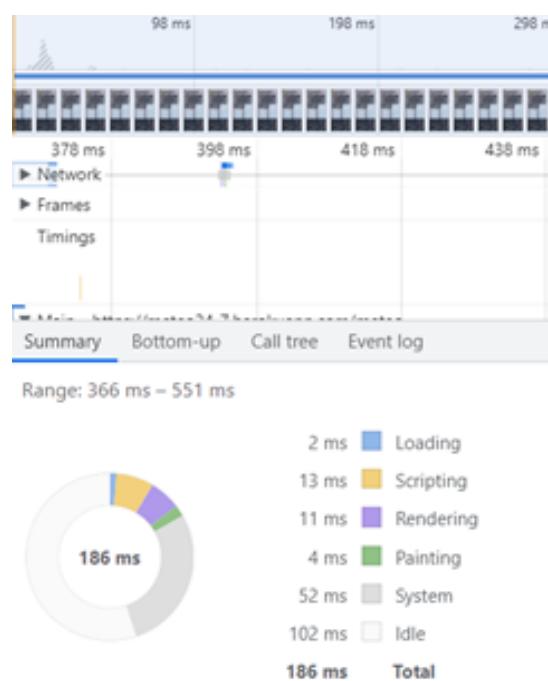
Grazie all'aiuto di strumenti offerti da google sotto ispeziona -> prestazioni è stato possibile osservare i tempi di loading dell'applicazione web e conoscere che aspetti fossero migliorabili.

Analisi iniziale:



Il risultato iniziale era migliorabile a livello di scripting, rendering e loading.

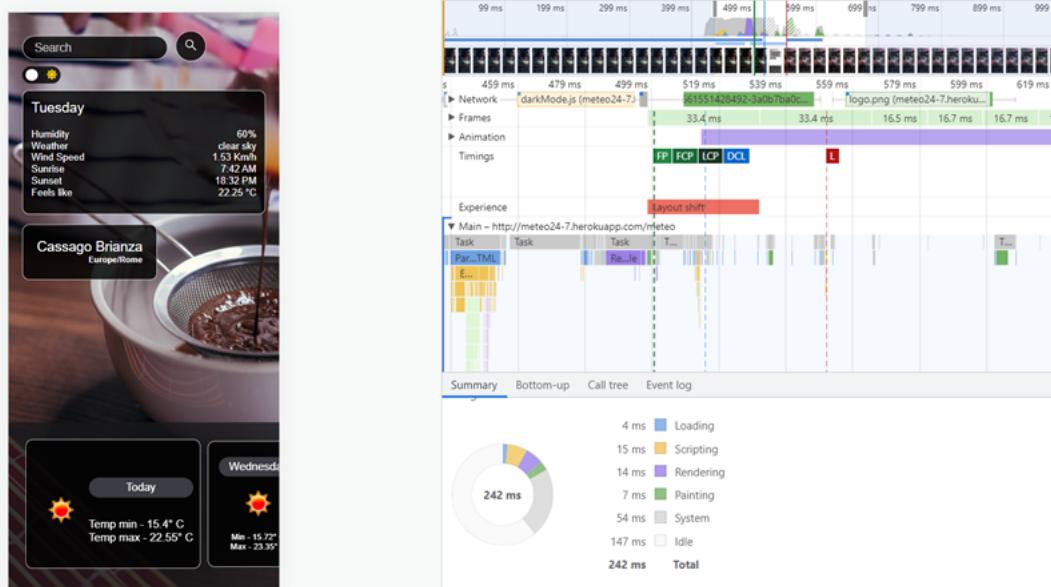
Risultato finale:



Come si è arrivati a questo risultato?

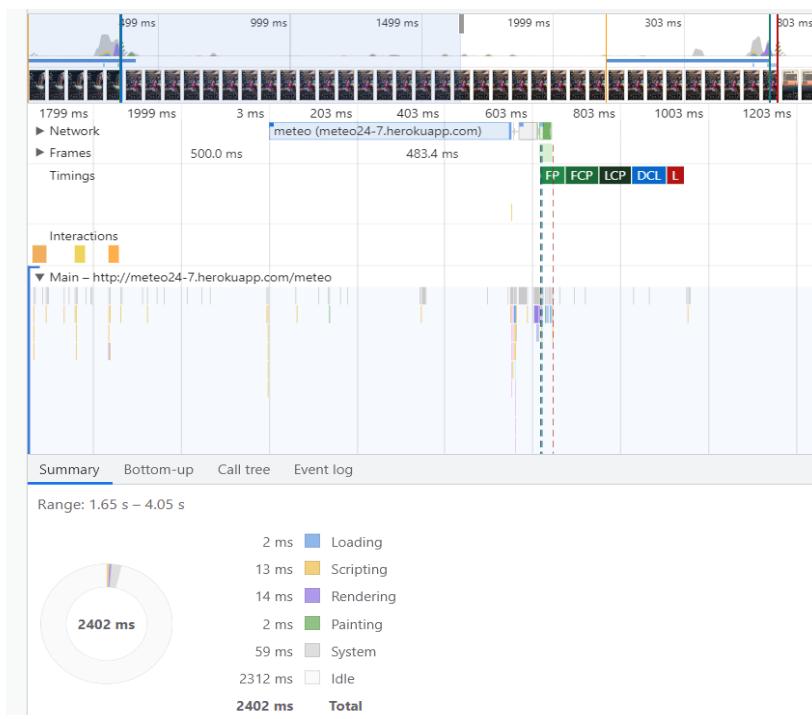
Essendo tutto gestito lato server (node.js), otteniamo un guadagno di prestazioni anche con le API call, sfruttando un approccio *non-blocking*, con una programmazione ad eventi. Inoltre, node.js, con il suo motore v8, garantisce elevate prestazioni.

(1) GET meteo.ejs



(1)

Un ulteriore guadagno di prestazioni, che si può notare nella foto(2) sottostante, è dovuto all'utilizzo del templating e data binding tramite EJS con approccio AJAX, e consiste in una riduzione dei tempi di painting, in quanto la pagina è già stata disegnata durante la richiesta GET, e successivamente, con la richiesta POST, viene aggiornata con i dati che cambiano.



(2)

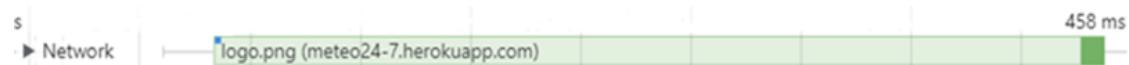
Un finale miglioramento è stato reso possibile tramite l'applicazione tecniche che sono state significative a livelli di tempo di caricamento e di prestazioni, ad esempio:

- lo spostamento degli script in fondo alla pagina, per ritardarne il caricamento e mostrare all'utente un caricamento più veloce
- la riduzione del codice css, tramite la eliminazione di codice inutilizzato, passando da questo:



a questo:

- Riduci i contenuti CSS inutilizzati
- utilizzo di una immagine webp per il logo, anziché jpg come era posta all'inizio, passando da questo:



a questo:



6. Usabilità

L'usabilità dell'applicazione è stata ottimizzata il più possibile data la sua importanza. Questo obiettivo è stato raggiunto rendendo visibili solo gli elementi utilizzabili, informazioni richieste e azioni che si possono svolgere per rendere il sistema il più naturale e fruibile.

E' responsive, quindi pensata per essere visualizzabile nel modo migliore per ogni tipo di dispositivo, qualsiasi sia la sua grandezza.

Tramite un approccio AJAX e templating è stato possibile aggiornare solamente i dati necessari e non l'intera pagina al momento di una ricerca da parte dell'utente.

Fondamentale è la separazione logica tra contenuto, struttura e stile per garantire un ciclo di vita indipendente dai dati.

7. Deploy sul web

Per pubblicare l'applicazione sul web, è stato usato il servizio sviluppato da *Salesforce* chiamato Herokuapp.

Questa piattaforma mette a disposizione un servizio gratuito che permette di fare il deploy sul web di una repository di github, fornendo un URI del tipo 'nomerepo.herokuapp.com', in questo caso 'meteo24-7.herokuapp.com'.

Fornisce inoltre una sistema di log in real time, utile per il debugging, oltre che strumenti di collaborazione e di team working.

8. Conclusioni

Meteo24-7 è un progetto nato nelle ore di laboratorio del corso di programmazione web, che poi ha preso vita grazie al suo sviluppo individuale.

Grazie ad esso è stato possibile mettere in pratica le informazioni acquisite durante le ore di lezione e quindi sviluppare finalmente un'applicazione web con le tecnologie presentate.

Gli obiettivi futuri riguardano i suoi possibili miglioramenti, sviluppando ad esempio una app per mobile che renda la sua consultazione più rapida e l'inserimento di nuove features come ad esempio una mappa corrispondente alla nazione della località cercata.

Infine si potrebbe pensare ad un eventuale guadagno monetario tramite l'inserimento di piccoli adv, migliorando la SEO, vertendo sulla quantità di traffico di utenza che utilizza l'applicazione.

9. Sitografia

<https://openweathermap.org/api>
<https://getbootstrap.com>
<https://unsplash.com>
<https://nodejs.org/it>
<https://expressjs.com/it>
<https://www.npmjs.com/package/dotenv>
<https://ejs.co>
<https://www.heroku.com>
<https://pagespeed.web.dev>
<https://www.lucarosati.it/blog/strategie-ricerca-informazione>
<https://schema.org>
https://validator.w3.org/#validate_by_input
<https://uncss-online.com>
<https://caniuse.com>
<https://github.com/retirejs/retire.js>
<https://leafletjs.com>
https://www.canva.com/it_it