Corso di Programmazione Web e Mobile

A.A. 2021-22

UniMeteo | Il tuo servizio meteorologico online!

Omar Daniel, 01664A

Autore: Omar Daniel
Ultima modifica: 20 luglio 2022 – versione 2.0
Prima modifica: 5 luglio 2022

Indice

Introduzione

UniMeteo è un'applicazione meteo che permette di ottenere informazioni climatiche di qualsiasi località sul globo. Sono presenti informazioni come temperatura, clima, temperatura percepita, umidità, velocità del vento, alba e tramonto (OpenWeather API). Inoltre, si può trovare un indice per la qualità dell'aria (AirPollution API) e le condizioni climatiche (descrizione, temperatura minima e massima) per la giornata corrente e i seguenti cinque giorni. Presenta una pagina iniziale che funge da landing page (con relativo collegamento con profilo Linkedin e GitHub contenente il codice sorgente) e una pagina contenente le informazioni sulla città richiesta (templating con EJS)

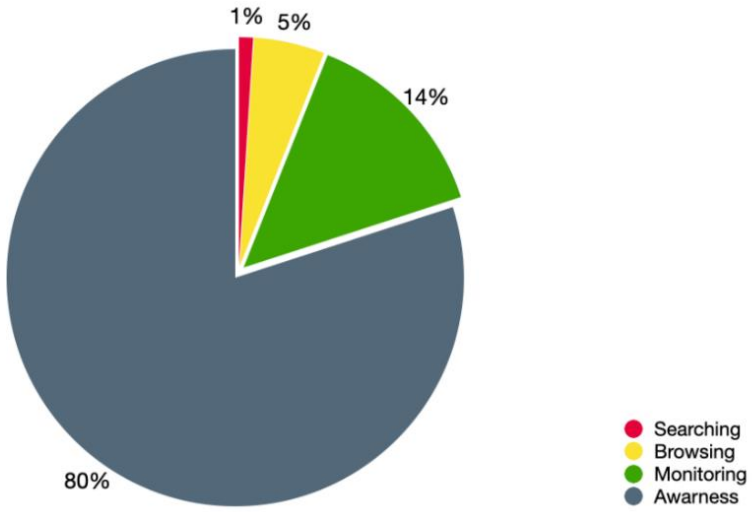
Breve analisi dei requisiti

Destinatari

La seguente applicazione web è pensata per essere utilizzata dalle persone che vogliono informarsi sul meteo in una certa località per motivi di viaggio o per interesse personale. Il contenuto viene presentato in inglese, permettendo un afflusso maggiore di utenti (le informazioni principali possono essere comprese anche da chi parla e capisce una lingua diversa). L'ipotetico destinatario non necessita di particolari conoscenze informatiche o scientifiche per l'utilizzazione del servizio, conosce già cosa vuole cercare e ha un livello di motivazione medio, caratterizzato da curiosità e necessità. La landing page serve, oltre che come pagina di introduzione, come pagina di condivisione che contiene una prima presentazione del servizio con un bottone per accedervi. Essendo un'applicazione caratterizzata da un design responsivo, gli utenti destinatari possono visitare il sito web utilizzando qualsiasi dispositivo elettronico, come computer, tablet e smartphone.

E' 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:

- Directed – Active → searching, quindi nel caso di UniMeteo, quando si ricerca una città per vederne il clima per un eventuale viaggio programmato nei giorni immediatamente successivi.
- Directed – Passive → monitoring, nel caso di UniMeteo, quando si apre la pagina meteo solamente per controllare giornalmente la temperatura della città in cui ci si trova
- Undirected – Active → browsing, tramite l'indicizzazione delle pagine dai motori di ricerca e la presenza di una landing page, nel caso in cui venga cercato un servizio meteo, UniMeteo potrebbe apparire come risultato della ricerca (possibilità di acquisire nuove informazioni)



- Undirected – Passive → awarness, nel caso di un'acquisizione passiva e indiretta delle informazioni, ad esempio tramite pubblicità, passaparola e/o condivisione.

Interfacce

L'applicazione presenta due interfacce:

1. Index → è una pagina di presentazione del progetto, con un bottone per accedere al servizio meteo e un collegamento diretto al profilo Linkedin e GitHub nel footer della pagina. Presenta alcune animazioni per attirare l'attenzione dell'utente sugli elementi principali che la costituiscono, come il titolo e il bottone.
2. Meteo → è la pagina che permette di usufruire del servizio, dando la possibilità all'utente di cercare una città e vederne tutte le informazioni relative. Presenta una sorta di "card" contenente le informazioni, impostando il focus dell'utente proprio su di essa, senza che ci siano interferenze inutili a distrarlo. È presente un input che restituisce un output

e che guidano quindi la logica dell'architettura. Una volta che l'utente inserisce e conferma il suo input, vengono generate alcune call API che restituiscono i dati richiesti, i quali vengono impaginati. Si possono effettuare richieste con metodo GET o POST. Il campo di input è impostato come required, cioè obbligatorio da inserire

Le interfacce sono state inizialmente progettate, cercando di ottenere il miglior risultato a livello di stile comunicativo, e successivamente realizzate mediante HTML5 (HyperText Markup Language) e CSS3 (Cascading Style Sheet).

Usabilità

Per favorire l'usabilità utente dell'applicazione, si sono resi visibili solo gli elementi utilizzabili e le azioni che si possono svolgere e si è cercato di rendere il sistema il più naturale e familiare possibile. Inoltre, è stata pensata per essere accessibile a qualsiasi tipo di utente, da qualsiasi dispositivo. Utilizzando un approccio AJAX (Asynchronous JavaScript and XML) tramite templating e data binding, si è evitato il refresh della pagina, considerato disorientante e destabilizzante per l'utente, ma vengono solamente aggiornati i dati necessari. Inoltre, grazie allo stesso approccio, è stato possibile garantire un ciclo di vita indipendente ai dati, separando logicamente struttura, stile e contenuto.

Flusso dei dati

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 (client) e server avviene nel momento della ricerca della città desiderata, 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 di una richiesta POST accedendo al body della richiesta (e' una funzione di middleware, cioè intermedia che viene effettuata tra la richiesta e la risposta: `app.use(express.urlencoded({ extended: true }))`). I dati vengono recuperati dal servizio OpenWeather che mette a disposizione delle API. Le call API vengono

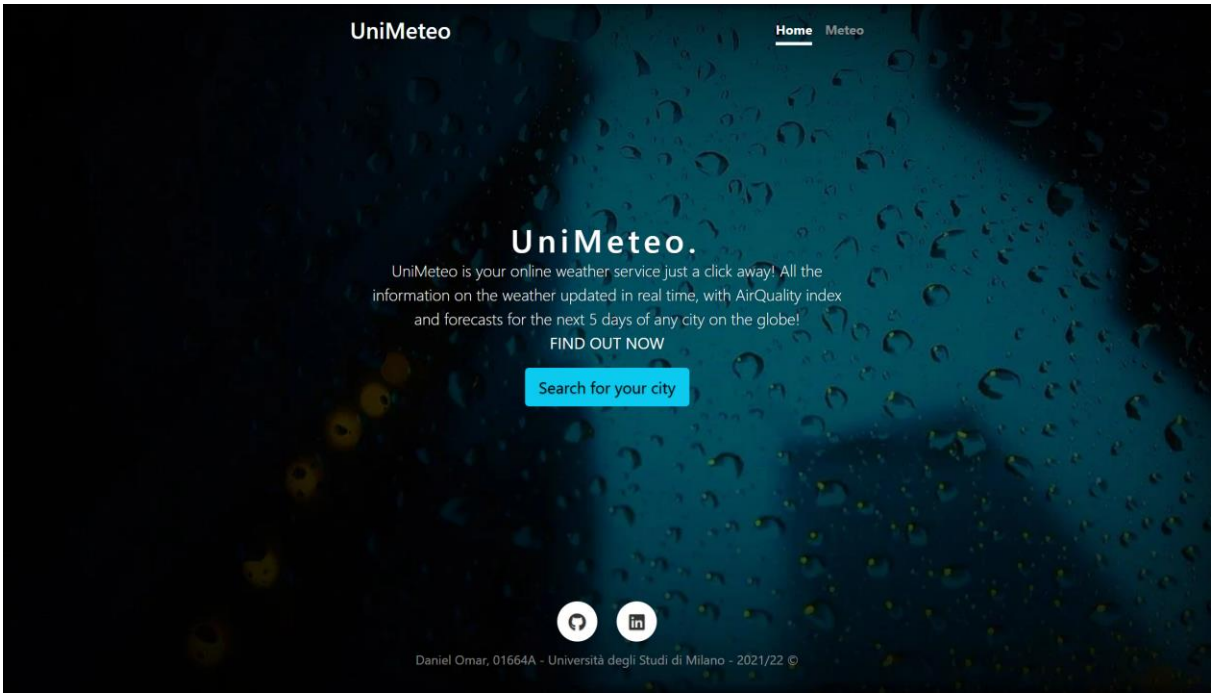
effettuate nel file *routes.js*. Per lo sfondo, invece, viene utilizzata una call API al servizio di Unsplash, il quale restituisce una serie di immagini relative alla query effettuata. Quindi tutti i contenuti sono reperiti online e a costo zero, anche di mantenimento.

Modello di valore

La seguente applicazione non è a scopo di lucro, ma può essere considerata un esempio pratico del fatto che per inserirsi nel mondo dell'web e dell'economia dell'informazione non si necessitano ingenti patrimoni. UniMeteo è stato ideato per essere gratuito e senza pubblicità invasiva ed eccessiva. Un modo per ottenere dei guadagni da quest'applicazione, sarebbe inserire delle ads e in questo caso il valore del rendimento dipenderà dal volume del traffico generato dal sito e dalla capacità di profilazione degli utenti.

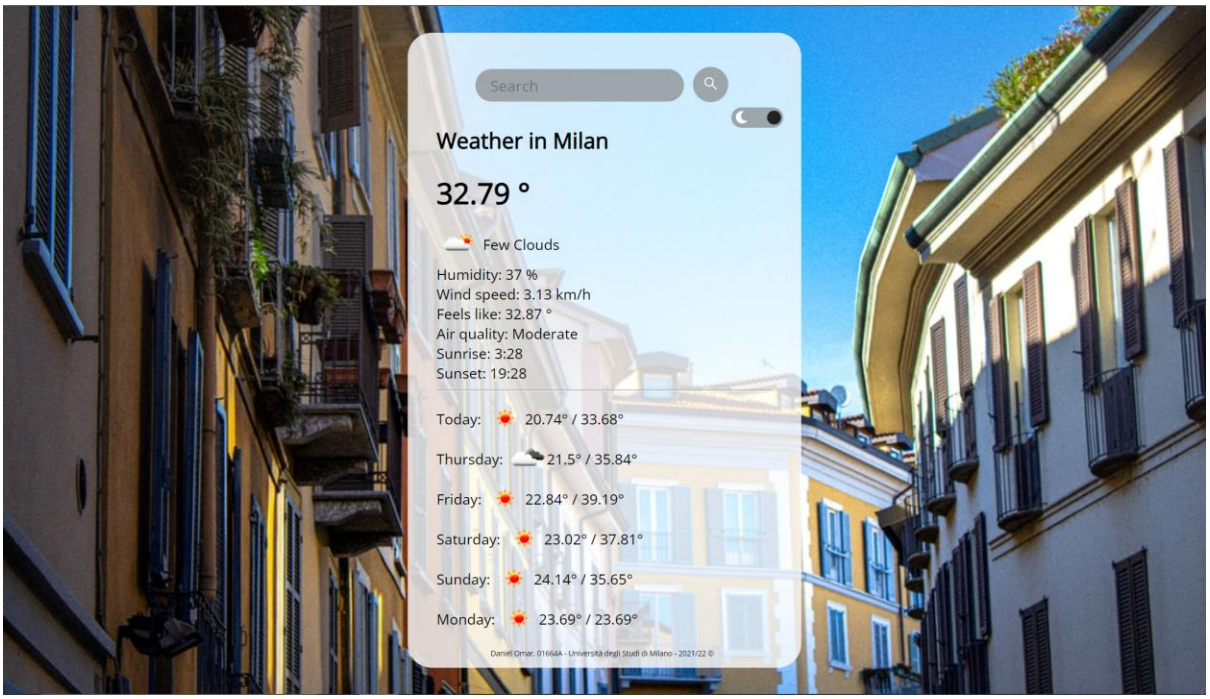
Panoramica interfacce utente

Index.ejs



Meteo.ejs

LightMode



DarkMode

Tecnologie utilizzate

HTML5

Con l'utilizzo di HTML5 sono state create le strutture per le due pagine e i relativi collegamenti ipertestuali presenti. Sono stati importati diversi file esterni, tra cui alcuni CSS per le icone o per il font, oltre a quelli interni che vanno a descrivere lo stile delle pagine (divisione struttura e stile). Tutte le pagine sono state validate tramite "The W3C Markup Validation Service"

CSS3

Grazie all'utilizzo di diversi file CSS, le pagine hanno assunto un design semplice, minimale ma anche animato, come il titolo e il bottone della pagina index.ejs. Inoltre, è stato possibile disporre gli elementi sulla pagina così come si possono vedere ora, cercando di spostare l'attenzione dell'utente e di mantenerla sul blocco contenente le informazioni del meteo. Sfruttando strumenti come "UnCSS", sono state rimossi i segmenti di codice non utilizzati e superflui

JavaScript

Il cuore dell'applicazione risiede in javascript, utilizzato sia lato client, ad esempio gli script per impostare light/dark mode e memorizzarlo nel local storage, sia lato server grazie all'utilizzo di Node.JS

Node.JS

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

- Express → per la gestione del routing, il deploying del server e la gestione delle views. All'interno del file *app.js* si trovano le funzioni di middleware e il deploy del server, mentre all'interno del file *routes.js* si può trovare la gestione della logica di routing per gestire tutti gli endpoint raggiungibili dall'utente e la renderizzazione delle views con i dati richiesti aggiornati
- Node-fetch → e' un modulo che permette l'uso di "fetch" lato server. Fetch, considerato l'evoluzione di XMLHttpRequest perché rende più semplice effettuare richieste asincrone e si gestiscono meglio le response, permette di effettuare le chiamate alle API utilizzate, ricevendo i dati come risposta. Funziona in modo asincrono, cioè le variabili che conterranno la risposta vengono inizializzate solamente quando la risposta è pronta, nel frattempo conterranno una Promise. Ciò va indicato tramite async e await, così che l'esecuzione di quella funzione aspetti la risposta con i dati prima di proseguire.
- Dotenv → si occupa di caricare tutte le variabili di ambiente contenute in un file .env dentro process.env, come le API key e la porta del server in caso non ce ne siano altre disponibili. Questo permette anche una maggiore segretezza di queste ultime
- Ddos → che offre una copertura per attacchi di tipo DOS (Denial-Of-Service), restituendo un errore dopo un certo numero di richieste effettuate entro un certo limite di tempo

Local storage

In quest'applicazione viene sfruttato il local storage per memorizzare una stringa JSON del tipo { lightMode: true/false } che ha permesso, con l'ausilio di script javascript, di memorizzare la scelta dell'utente e caricarla ogni qual volta avvenga il refresh della pagina.

API

Le API utilizzate sono:

- OneCall API 1.0 by OpenWeather che restituisce, in formato JSON, tutti i dati relativi al meteo del giorno corrente e dei 7 giorni successivi. I dati utilizzati sono la temperatura, la velocità del vento, l'umidità, la temperatura percepita, la temperatura massima, quella minima, l'icona del clima, una descrizione del cielo, l'orario dell'alba e del tramonto.
- Geocoding API by OpenWeather che permette, a partire dal nome di una città, di ricevere latitudine e longitudine corrispondenti. Necessario in quanto OneCallAPI 1.0 e AirPollution API le utilizzano
- AirPollution API by OpenWeather per avere un indice della qualità dell'aria (restituisce un indice numerico che viene trasformato poi in testuale seguendo l'apposita tabella descrittiva)
- Unsplash API che restituisce un elenco di immagini relative alla parola ricercata. Usata per avere un background diverso ogni volta che si effettua una ricerca
- IP-API che restituisce informazioni relative ad un indirizzo IP. In quest'applicazione web è stato usato per recuperare le coordinate dell'IP del client durante una richiesta GET a meteo.ejs per poi recuperare e mostrare i dati relativi alla località corrispondente.

EJS

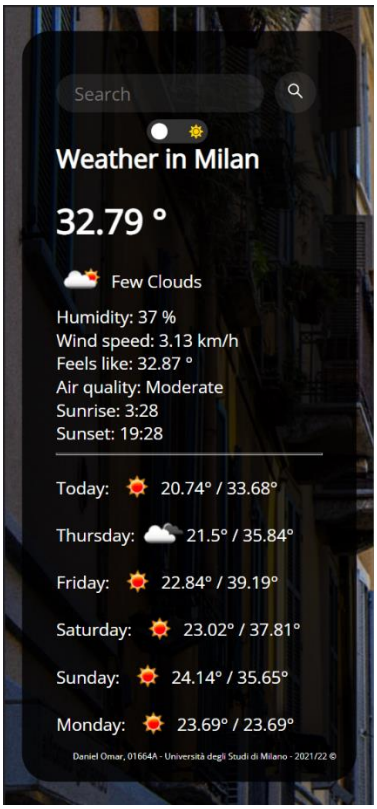
Il progetto sfrutta un semplice linguaggio di templating chiamato EJS (Embedded Javascript templates). Questo linguaggio ha permesso di creare pagine dinamiche che assumono aspetti diversi in base ai dati ricevuti. Era indispensabile utilizzare questa tecnologia in quanto bisogna mostrare dati diversi in base alla città cercata. Viene integrata direttamente all'interno dell'HTML con tag specificati come <%= %> o <% %>. I dati da renderizzare vengono specificati all'interno di *routes.js.*

Herokuapp

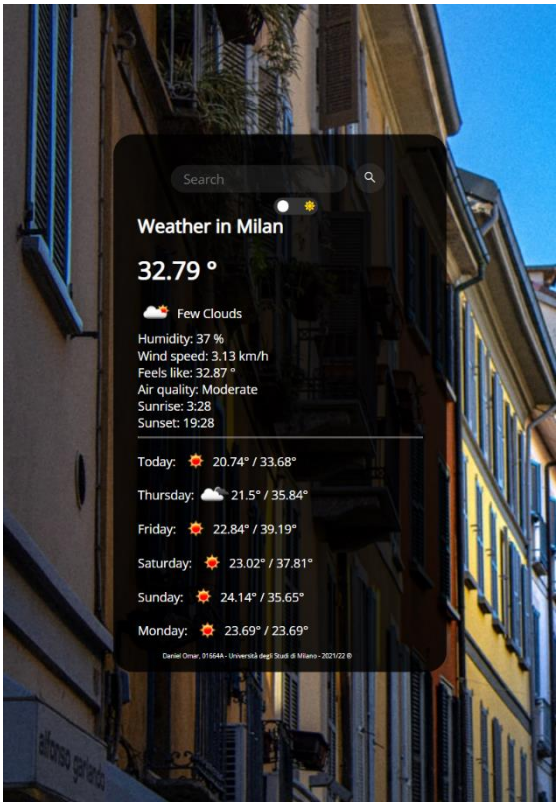
Herokuapp è l'architettura che ha permesso l'upload dell'applicazione web su internet, fornendo un URI raggiungibile da tutti i dispositivi. È un'architettura sviluppata da Salesforce che permette di fare il deploy sul web di una repository di GitHub e fornisce inoltre un sistema di log in tempo reale (ha permesso di evitare l'uso del modulo node.js "morgan"). In questo caso, l'URI diretto al sito web è unimeteo.herokuapp.com

Responsive design

Tutte le pagine hanno un design responsivo, ottenuto tramite librerie (come Bootstrap) oppure tramite CSS.

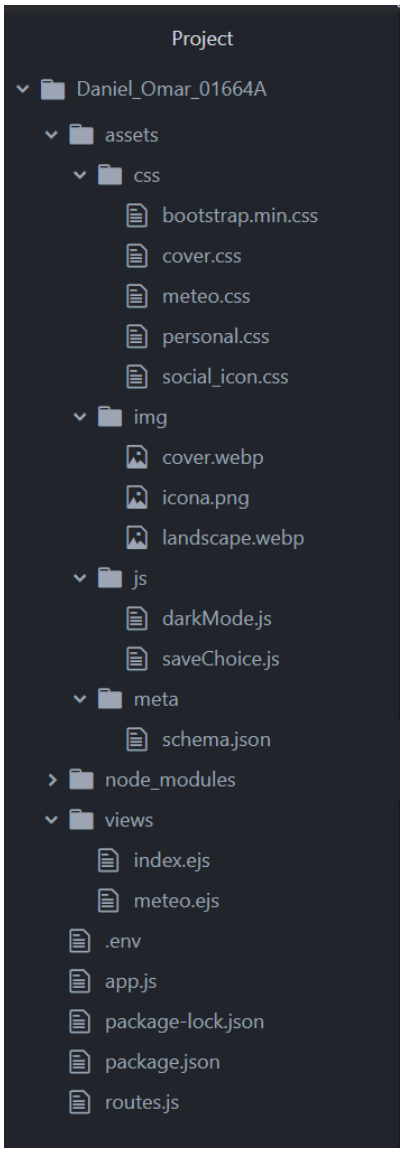


Iphone 12 pro



Ipad Air

Architettura



Nella cartella assets troviamo delle sub-directory contenenti i fogli di stile, gli script javascript, le immagini per il sito web e uno schema JSON contenente i metadati per l'indicizzazione della pagina dai motori di ricerca (creato seguendo lo schema "WebPage" presente su schema.org).

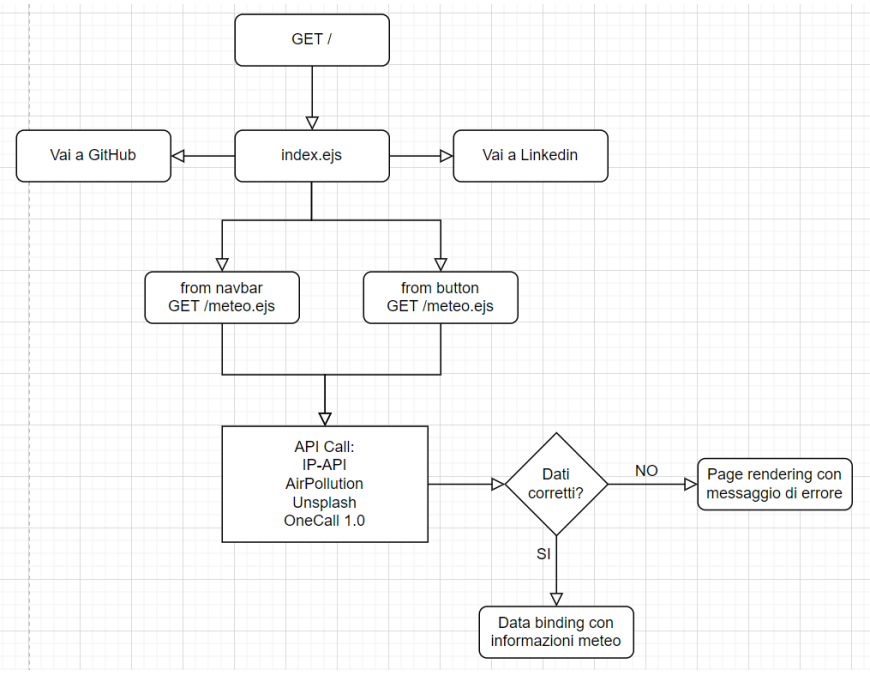
All'interno della cartella views si trovano le due interfacce utente, index.ejs e meteo.ejs, templates in cui avverrà poi il data binding con le informazioni della città richiesta.

Package.json e package-lock.json contengono tutte le dependecies di node.js, cioè tutti i moduli necessari per il funzionamento. Inoltre, la cartella node_modules contiene le installazioni di tutti i pacchetti richiesti.

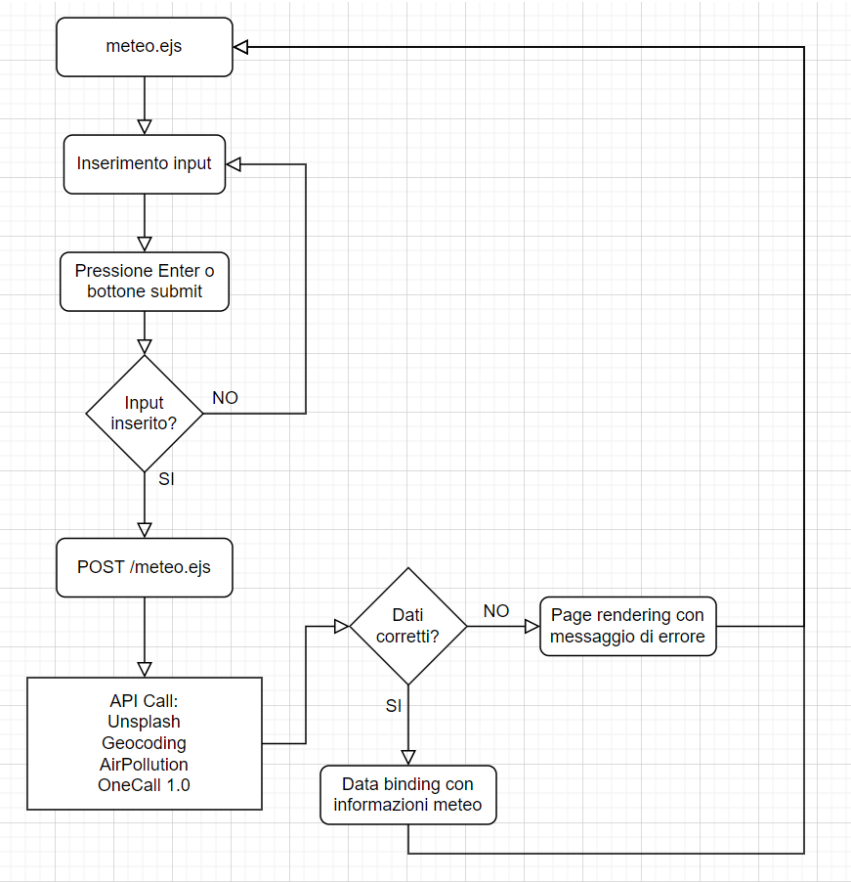
Nel file .env si trovano tutte le variabili di ambiente, come le API key e il numero di porta del server.

Nel file app.js si trova la configurazione di un server express, con relative funzioni di middleware, mentre nel file routes.js si trova la gestione del routing e quindi degli endpoint raggiungibili dall'utente, con il rendering delle views e l'impostazione delle variabili EJS.

Funzionamento index.ejs



Funzionamento meteo.ejs



Sezioni del codice (IDE utilizzato: atom.io)

Index.ejs

```
<header class="mb-auto">
  <div>
    <h3 class="float-md-start mb-0">UniMeteo</h3>
    <!-- Nav bar con collegamenti alle pagine -->
    <nav class="nav nav-masthead justify-content-center float-md-end">
      <a class="nav-link fw-bold py-1 px-0 active" aria-current="page" href="/">Home</a>
      <a class="nav-link fw-bold py-1 px-0" href="/meteo">Meteo</a>
    </nav>
  </div>
</header>
<!-- Main della pagina, con titolo, descrizione e bottone -->
<main class="px-3">
  <h1 class="typewriter">UniMeteo.</h1>
  <p class="lead">UniMeteo is your online weather service just a click away! All the information on the weather updated in real time, with AirQuality index and forecasts
    for the next 5 days of any city on the globe!<br><b>FIND OUT NOW</b></p>
  <p class="lead">
    <a href="/meteo" class="btn btn-lg btn-info animate__animated animate__jackInTheBox">Search for your city</a>
  </p>
</main>
<!-- Footer con indicazione tramite bottone a social network -->
<footer class="mt-auto text-white-50">
  <div class="wrapper">
    <a href="https://github.com/omarrdaniel" class="icon github" target="_blank">
      <div class="tooltip">Github</div>
      <span><i class="fab fa-github"></i></span>
    </a>
    <a href="https://www.linkedin.com/in/omarrdaniel/" class="icon linkedin" target="_blank">
      <div class="tooltip">Linkedin</div>
      <span><i class="fab fa-linkedin"></i></span>
    </a>
  </div>
  <p>Daniel Omar, 01664A - Università degli Studi di Milano - 2021/22 &copy;</p>
</footer>
```

Pagina principale index.ejs con navbar, testo centrale, bottone e footer con icone social.

Meteo.ejs

```
37        <div class="weather">
38          <% if(city !== null) { %>
39            <h2 class="city">Weather in <%= city %></h2>
40          <% } %>
41          <% if(temp !== null) { %>
42            <h1 class="temp"><%= temp %> °</h1>
43          <% } %>
44          <div class="flex">
45            <% if(imgsrc !== null) { %>
46              <img src="<%= imgsrc %>" alt="icona meteo" class="icon" loadin="lazy">
47            <% } %>
48            <% if(description !== null) { %>
49              <div class="description"><%= description %></div>
50            <% } %>
51          </div>
52          <% if(humidity !== null) { %>
53            <div class="humidity">Humidity: <%= humidity %> %</div>
54          <% } %>
55          <% if(wind !== null) { %>
56            <div class="wind">Wind speed: <%= wind %> km/h</div>
57          <% } %>
58          <% if(feels !== null) { %>
59            <div class="feels">Feels like: <%= feels %> °</div>
60          <% } %>
61          <% if(aq !== null) { %>
62            <div class="aq">Air quality: <%= aq %></div>
63          <% } %>
64          <% if(sunrise !== null) { %>
65            <div class="sunrise">Sunrise: <%= sunrise %></div>
66          <% } %>
67          <% if(sunset !== null) { %>
68            <div class="sunset">Sunset: <%= sunset %></div>
69          <% } %>
70        </div>
71        <hr/>
72        <% if (min !== null && max !== null && imgtoday !== null) { %>
73          <div class="days">
74            <div class="flex">
75              <div>Today: </div>
76              <img src="<%= imgtoday %>" class="icon" loading="lazy" alt="icona meteo">
77              <div><%= min %>° / <%= max %>°</div>
78            </div>
79            <div class="flex">
80              <div><%= day1 %>: </div>
```

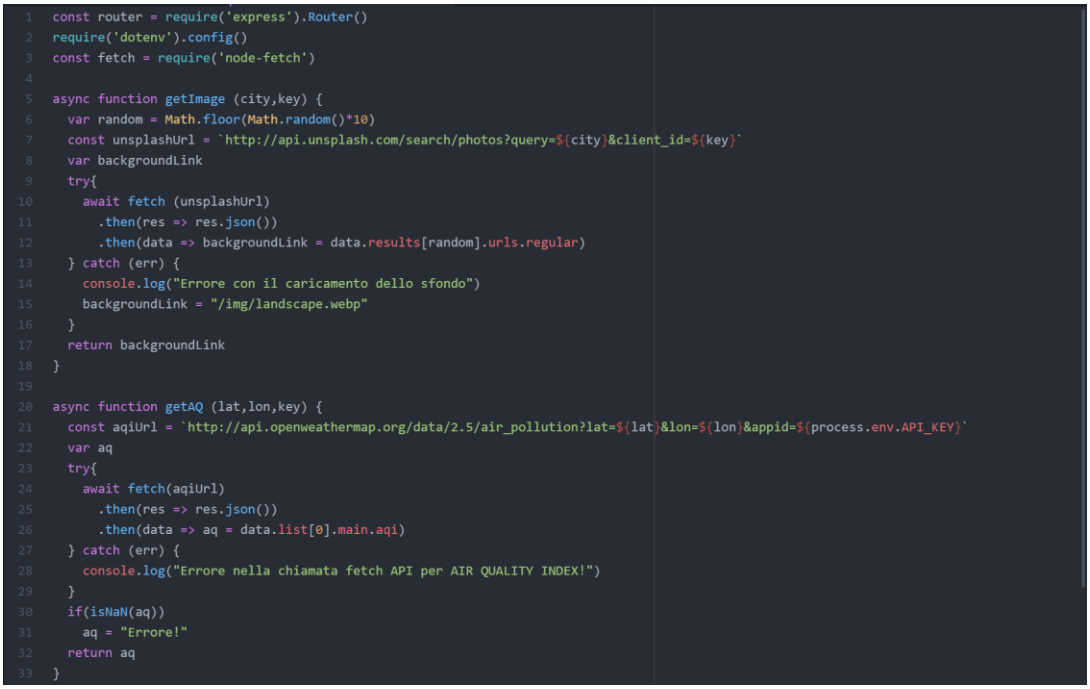
Gestione del data binding con EJS, con relative condizioni. In figura si può vedere la “card” del meteo con tutti i dati che verranno completati.

App.js

```
const express = require('express')
const Ddos = require('ddos')
const app = express()
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}`)
})
```

Deploying di un server express, con protezione anti-dos, funzioni di middleware e importing delle routes (gestite su file a parte)

Routes.js (1)



```
const router = require('express').Router()
require('dotenv').config()
const fetch = require('node-fetch')

async function getImage (city,key) {
  var random = Math.floor(Math.random()*10)
  const unsplashUrl = `http://api.unsplash.com/search/photos?query=${city}&client_id=${key}`
  var backgroundLink
  try{
    await fetch (unsplashUrl)
      .then(res => res.json())
      .then(data => backgroundLink = data.results[random].urls.regular)
  } catch (err) {
    console.log("Errore con il caricamento dello sfondo")
    backgroundLink = "/img/landscape.webp"
  }
  return backgroundLink
}

async function getAQ (lat,lon,key) {
  const aqiUrl = `http://api.openweathermap.org/data/2.5/air_pollution?lat=${lat}&lon=${lon}&appid=${process.env.API_KEY}`
  var aq
  try{
    await fetch(aqiUrl)
      .then(res => res.json())
      .then(data => aq = data.list[0].main.aqi)
  } catch (err) {
    console.log("Errore nella chiamata fetch API per AIR QUALITY INDEX!")
  }
  if(isNaN(aq))
    aq = "Errore!"
  return aq
}
```

Dichiarazione di alcune funzioni per una maggiore modularità e manutenibilità

Routes.js (2) – POST /meteo.ejs con rendering pagina

```
            var sunset = new Date (data.current.sunset * 1000)
            if(sunset.getMinutes()<10){
              var sunsmin = '0' + sunset.getMinutes()
            } else {
              var sunsmin = sunset.getMinutes()
            }
            res.render('meteo', {
              city: city,
              temp: data.current.temp,
              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,
              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",
              mintom: data.daily[1].temp.min,
              maxtom: data.daily[1].temp.max,
              imgdayafter: "https://openweathermap.org/img/w/" + data.daily[2].weather[0].icon + ".png",
              mindayafter: data.daily[2].temp.min,
              maxdayafter: data.daily[2].temp.max,
              unsplash: backgroundLink,
              day1: days[(date.getDay() + 1) % 7],
              day2: days[(date.getDay() + 2) % 7],
              day3: days[(date.getDay() + 3) % 7],
              imgday3: "https://openweathermap.org/img/w/" + data.daily[3].weather[0].icon + ".png",
              min3: data.daily[3].temp.min,
              max3: data.daily[3].temp.max,
              feels: data.current.feels_like,
              day4: days[(date.getDay() + 4) % 7],
              imgday4: "https://openweathermap.org/img/w/" + data.daily[4].weather[0].icon + ".png",
              min4: data.daily[4].temp.min,
              max4: data.daily[4].temp.max,
              day5:  days[(date.getDay() + 5) % 7],
              imgday5: "https://openweathermap.org/img/w/" + data.daily[5].weather[0].icon + ".png",
              min5: data.daily[5].temp.min,
              max5: data.daily[5].temp.max,
              sunrise: (sunrise.getHours()+2) + ":" + sunrmin,
              sunset: (sunset.getHours()+2) + ":" + sunsmin
            })
          }
        })
    } catch (err) {
      console.log("Errore nel Weather API Call")
```

Darkmode.js – con relative funzioni per impostare lightmode e darkmode

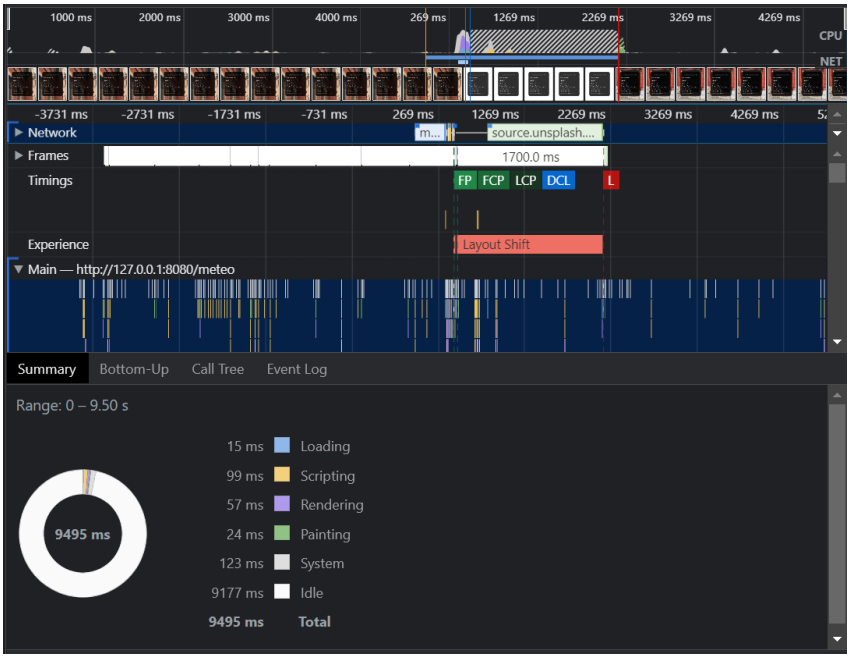
```
function setLight () {
  document.getElementById('card').style.backgroundColor = "rgba(255, 255, 255, 0.8)"
  document.getElementById('card').style.color = "black"
  document.getElementById('search-bar').style.backgroundColor = "rgba(135, 135, 135, 0.7)"
  document.getElementById('search').style.backgroundColor = "rgba(135, 135, 135, 0.7)"
}

function setDark () {
  document.getElementById('card').style.backgroundColor = "#000000d0"
  document.getElementById('card').style.color = "white"
  document.getElementById('search-bar').style.backgroundColor = "#7c7c7c2b"
  document.getElementById('search').style.backgroundColor = "#7c7c7c2b"
}

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

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

Savechoice.js – salvare scelta di light/dark mode in local storage

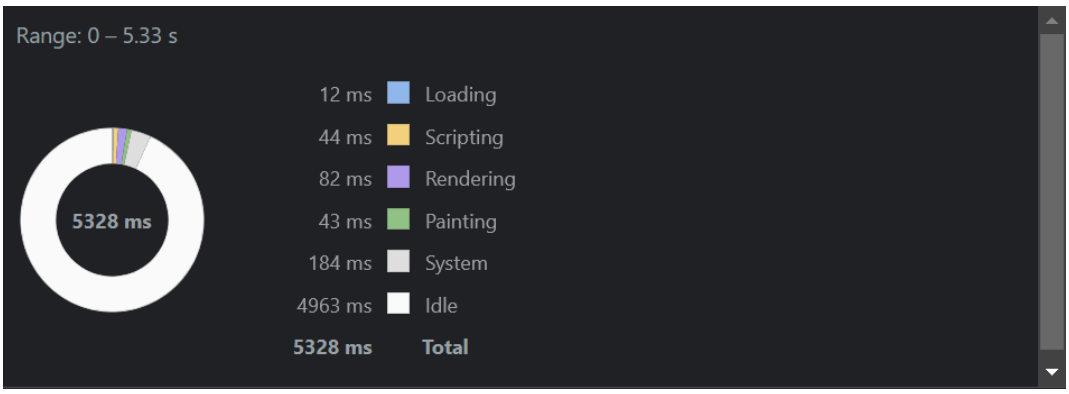
```
var scelta = document.getElementById("switch")

function setLight () {
  document.getElementById('card').style.backgroundColor = "rgba(255, 255, 255, 0.8)"
  document.getElementById('card').style.color = "black"
  document.getElementById('search-bar').style.backgroundColor = "rgba(135, 135, 135, 0.7)"
  document.getElementById('search').style.backgroundColor = "rgba(135, 135, 135, 0.7)"
}

function setDark () {
  document.getElementById('card').style.backgroundColor = "#000000d0"
  document.getElementById('card').style.color = "white"
  document.getElementById('search-bar').style.backgroundColor = "#7c7c7c2b"
  document.getElementById('search').style.backgroundColor = "#7c7c7c2b"
}

document.addEventListener('DOMContentLoaded', function () {
  //funzione per leggere il json e impostare switch dark mode
  const choice = JSON.parse(localStorage.getItem('mode'))
  if(choice.lightMode){
    scelta.checked=true
    setLight()
  } else {
    scelta.checked=false
    setDark()
  }
});

scelta.addEventListener('change', function () {
  if(scelta.checked){
    const obj = {
      lightMode: true,
    }
    localStorage.setItem('mode', JSON.stringify(obj));
  } else {
    const obj = {
      lightMode: false,
    }
    localStorage.setItem('mode', JSON.stringify(obj));
  }
});
```

Prestazioni

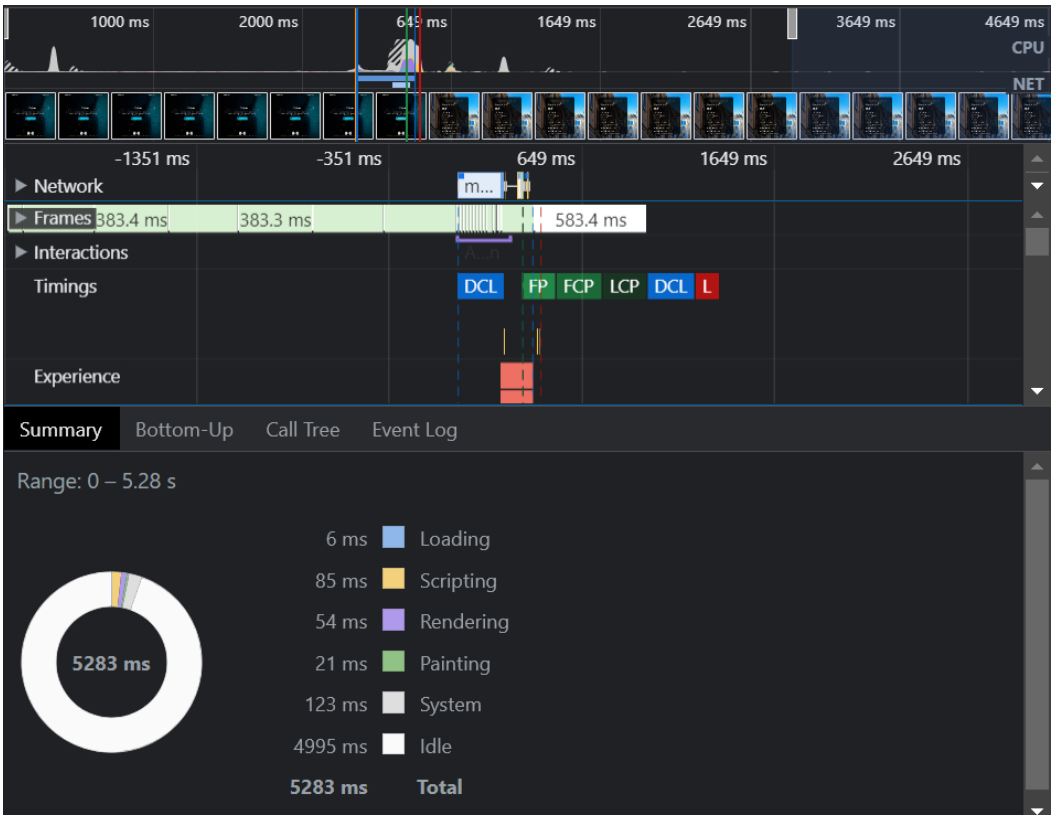
Inizialmente, per impostare un background casuale per ogni richiesta, veniva usato source.unsplash, un'alternativa alla normale API call. Visti i lunghi tempi di caricamento (circa 3s), si è passati all'utilizzo della comune call API al servizio di unsplash, notando un forte guadagno di prestazione nel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



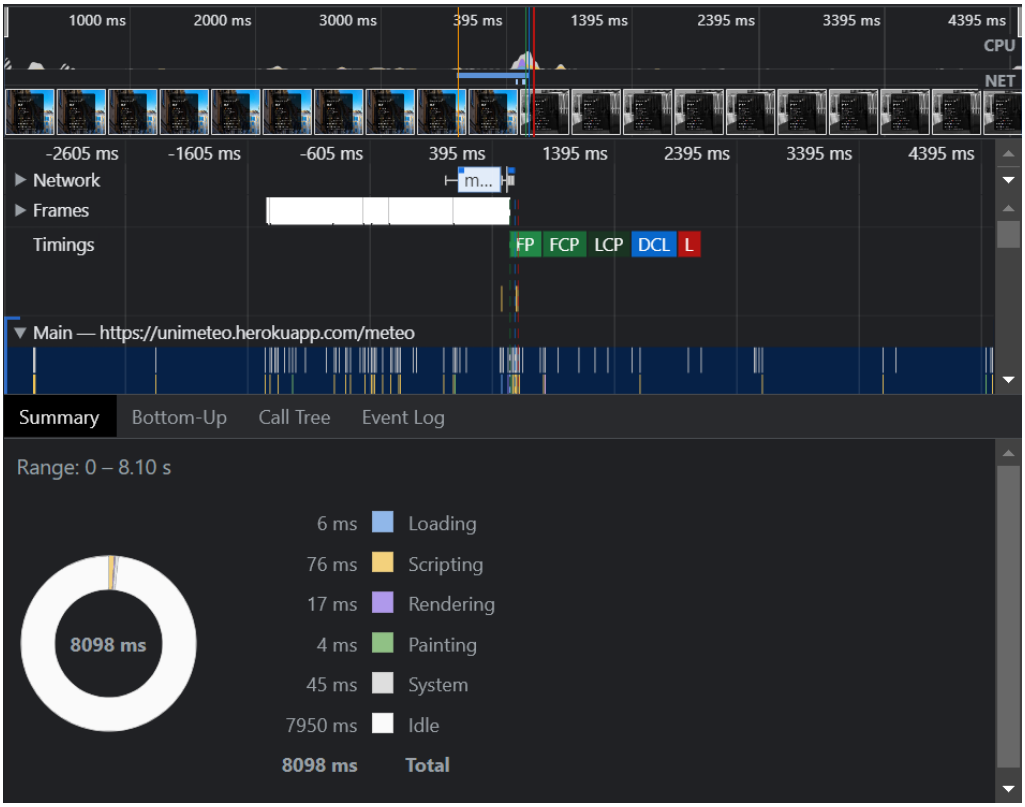
Il caricamento della pagina index.ejs risulta molto veloce, essendo principalmente statica. E' stato migliorato passando da uno sfondo .jpg (più pesante e lento) ad uno sfondo .webp (formato più leggero e versatile)



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. (Nella foto GET meteo.ejs)



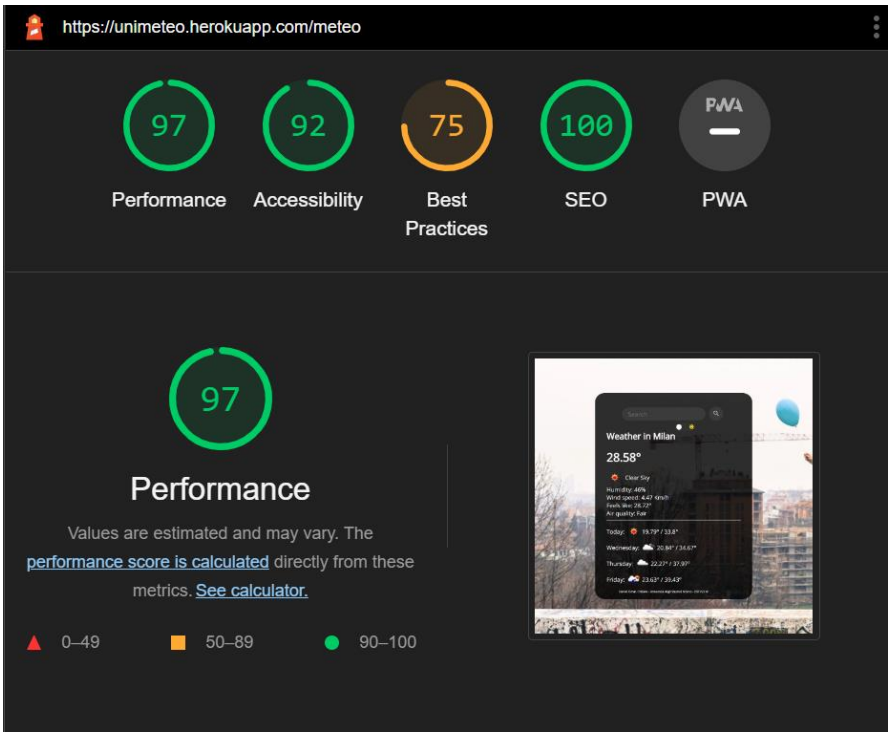
Lo stesso vale per una qualsiasi richiesta POST a meteo.ejs per cercare il meteo di qualunque città. Un ulteriore guadagno di prestazioni che si può notare nella foto sottostante, dovuto all'utilizzo del templating e data binding tramite EJS con approccio AJAX, è un quasi azzeramento dei tempi di rendering e painting, in quanto la pagina è già stata disegnata durante la richiesta GET, e successivamente, con la richiesta POST, vengono solo aggiornati i dati che cambiano.



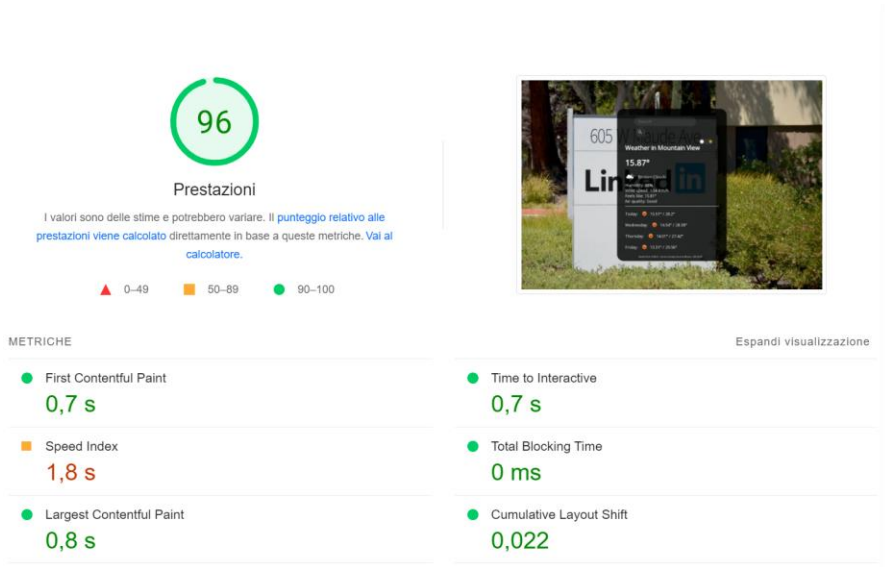
Un ulteriore miglioramento delle prestazioni è dovuto all'impiego del lazy loading per il caricamento delle immagini (nelle ultime versioni di Chrome aggiunto come attributo del tag img), all'inserimento degli script javascript in fondo alla pagina, così da ritardarne il caricamento e mostrare all'utente un caricamento più veloce. Poi, sfruttando strumenti come UnCSS, il codice è stato ridotto, rendendo i file più leggeri e di conseguenza più veloce l'applicazione web. Inoltre, facendo l'inlining del background della pagina, non bisogna aspettare il caricamento del css per impostarlo.

Si potrebbe ottenere un ulteriore miglioramento delle prestazioni tramite la riduzione al minimo del codice non utilizzato, eliminando i commenti e gli spazi vuoti, evitando l'utilizzo di librerie/CSS superflue, aumentando la fluidità e l'usabilità.

Risultati del lighthouse report by Google Chrome:



Risultati del report di PageSpeedInsight (utilizzabile in quanto l'applicazione è stata caricata sul web)



Sicurezza

Sono state introdotte delle politiche di cybersecurity, come una protezione anti-dos, gestita dal modulo ddos di Node.JS: la prima richiesta arriva e la scadenza è fissata a 1 secondo. Se trascorre 1 secondo e non vengono effettuate richieste aggiuntive, la voce viene rimossa dalla tabella interna. Infatti, può esserci un numero massimo di richieste effettuate e il tempo di scadenza non cambierà. L'unico modo in cui la scadenza aumenta è quando arriva una richiesta, il conteggio aumenta; quindi, se il conteggio supera l'importo del burst, la scadenza sale al doppio del valore precedente. (*burst* indica il numero di richieste oltre il quale il client viene penalizzato e la expiration incrementa del doppio di quella precedente, mentre *limit* indica il numero di richieste oltre il quale le richieste vengono negate). Inoltre, gestendo le call API lato server, tutte le API key sono nascoste ad un eventuale utente malintenzionato (nel caso di api che trattano dati sensibili o a pagamento), nonché tutte le funzioni e la logica di funzionamento dell'applicazione web.

(screen dei log in caso di tentato attacco dos simulato)

```
Safari/537.36 { count: 56, expiry: 120 }
2022-07-11T18:14:14.982893+00:00 app[api]: Deploy 8f17b2f2 by user omar.daniel@studenti.unimi.it
2022-07-11T18:14:43.185519+00:00 heroku[router]: at=info method=GET path="/" host=unimeteo.herokuapp.com request_id=4a7f386d-ca7b-408c-ad0a-5b328cd35561 fwd="5.95.64.167"
dyno=web.1 connect=0ms service=1ms status=429 bytes=188 protocol=https
2022-07-11T18:14:42.547926+00:00 heroku[router]: at=info method=GET path="/" host=unimeteo.herokuapp.com request_id=fc597835-f27b-4201-a26d-266507e22613 fwd="5.95.64.167"
dyno=web.1 connect=0ms service=1ms status=429 bytes=188 protocol=https
2022-07-11T18:14:43.319876+00:00 heroku[router]: at=info method=GET path="/" host=unimeteo.herokuapp.com request_id=73a493d1-0e81-4a9f-a956-ad86f2bd3138 fwd="5.95.64.167"
dyno=web.1 connect=0ms service=1ms status=429 bytes=188 protocol=https
2022-07-11T18:14:43.473265+00:00 heroku[router]: at=info method=GET path="/" host=unimeteo.herokuapp.com request_id=4ba1b1e4-9d7b-40ef-8d72-b0b0a9f546cb fwd="5.95.64.167"
dyno=web.1 connect=0ms service=1ms status=429 bytes=188 protocol=https
2022-07-11T18:14:41.563249+00:00 app[web.1]: ddos: denied: entry: 5.95.64.167#Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0
Safari/537.36 { count: 52, expiry: 120 }
2022-07-11T18:14:41.539502+00:00 heroku[router]: at=info method=GET path="/" host=unimeteo.herokuapp.com request_id=7c65f6a5-869c-4893-86c6-1d4d6010a4d6 fwd="5.95.64.167"
dyno=web.1 connect=0ms service=1ms status=429 bytes=188 protocol=https
2022-07-11T18:14:43.209324+00:00 app[web.1]: ddos: denied: entry: 5.95.64.167#Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0
Safari/537.36 { count: 57, expiry: 120 }
2022-07-11T18:14:43.343558+00:00 app[web.1]: ddos: denied: entry: 5.95.64.167#Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0
Safari/537.36 { count: 58, expiry: 120 }
2022-07-11T18:14:43.012478+00:00 heroku[router]: at=info method=GET path="/" host=unimeteo.herokuapp.com request_id=90bff3a4-f824-40ea-9c1e-8361bd3c3a08 fwd="5.95.64.167"
dyno=web.1 connect=0ms service=2ms status=429 bytes=188 protocol=https
2022-07-11T18:14:45.999023+00:00 app[web.1]: ddos: denied: entry: 5.95.64.167#Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0
Safari/537.36 { count: 60, expiry: 120 }
2022-07-11T18:14:43.497087+00:00 app[web.1]: ddos: denied: entry: 5.95.64.167#Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0
Safari/537.36 { count: 59, expiry: 120 }
2022-07-11T18:14:45.975548+00:00 heroku[router]: at=info method=GET path="/" host=unimeteo.herokuapp.com request_id=e1e71222-a54f-458a-b547-8637cdd8d15d fwd="5.95.64.167"
dyno=web.1 connect=0ms service=1ms status=429 bytes=188 protocol=https
```

Inoltre, sfruttando il servizio offerto da retire.js, è stato effettuato un controllo sulle dipendenze/librerie di node.js e sulle relative vulnerabilità note (CVE): non ne sono state rilevate. (nella foto un esempio di sito che presenta vulnerabilità

Deploy sul web

Per pubblicare l'applicazione sul web, è stato usato il servizio sviluppato da Salesforce chiamato Herokuapp. Questo servizio mette a disposizione un piano free che permette di fare il deploy di una repository di github, fornendo un URI del tipo nomerepository.herokuapp.com. Fornisce inoltre una sistema di log in real time, oltre che strumenti di collaborazione e di team working. Presenta anche dei piani a pagamento con feature aggiuntive.

Conclusioni

L'idea dell'applicazione web è stata presentata inizialmente durante le ore di laboratorio, poi sviluppata, realizzata e perfezionata individualmente. Quante volte capita che prima di partire per un viaggio ci si chieda come sarà il tempo nella città di destinazione anche per sapere come preparare la valigia? Ecco, UniMeteo risponde proprio a questo quesito, cioè di informare l'utente sulle condizioni meteorologiche di una certa città.

L'obiettivo futuro è di migliorare questo progetto, inserendo una mappa climatica basata su *leaflet*, tramite alcuni layer e dati forniti da API (anche con possibilità di API a pagamento che supportano la creazione di questa mappa), una futura app mobile per una consultazione ancora più rapida e comoda e migliori prestazioni. Per permettere l'utilizzo di API a pagamento, potrebbero essere anche inserite piccole pubblicità non invadenti.

https://github.com/omarrdaniel/unimeteo

Sitografia

https://pagespeed.web.dev/

https://www.lucarosati.it/blog/strategie-ricerca-informazione consultato il 12/07/2022

https://www.heroku.com/

https://expressjs.com/it/

https://www.npmjs.com/package/ddos

https://ip-api.com/

https://openweathermap.org/api

https://unsplash.com/

https://nodejs.org/it/

https://www.npmjs.com/package/dotenv

https://getbootstrap.com/

https://schema.org/

https://validator.w3.org/#validate_by_input

https://uncss-online.com/

https://caniuse.com/

https://ejs.co/

https://github.com/retirejs/retire.js/ consultato il 13/07/2022

https://leafletjs.com/

https://www.canva.com/it_it/