Corso di Programmazione Web e Mobile

A.A. 2022-23

IlTuoMeteo | Il tuo servizio meteorologico online!

Domenico Marrani

976796

Autore :Domenico Marrani
MATR:976796

Indice

Introduzione 2

Breve analisi dei requisiti 2
- **Destinatari** 2

Interfacce 3

Usabilità 4

Flusso dei dati 4

Modello di valore 5

Panoramica Interfacce Utente 5
- Index.ejs 5
- Meteo.ejs 6
  - LightMode 6
  - DarkMode 6

Tecnologie utilizzate 7
- HTML5 7
- CSS3 7
- JavaScript 7
- Node.JS 7
- Local Storage 8
- API 8
- EJS 9
- Responsive Design 9

Architettura delle risorse 10

Funzionamento di Index.ejs 11

Funzionamento di Meteo.ejs 11

Sezioni del codice (IDE utilizzato: vs studio code) 12
- Index.ejs 12
- Meteo.ejs 13
- App.js 14
- Route.js 15
- Darkmode.js – con relative funzioni per impostare lightmode e darkmode 18
- SaveChoice.js – salvare scelta di light/dark mode in local storage 18

Prestazioni 19
- Index.ejs 19
- Meteo.ejs 19
- Node.js 19

Sicurezza 20

Conclusioni 20

Introduzione

IlTuoMeteo è un'applicazione meteo che permette di ottenere informazioni climatiche di qualsiasi località del 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. Il progetto presenta una pagina iniziale che funge da landing page (con relativi collegamenti ai profili GitHub di entrambi gli studenti) e una pagina contenente le informazioni sulla città richiesta (templating con EJS)

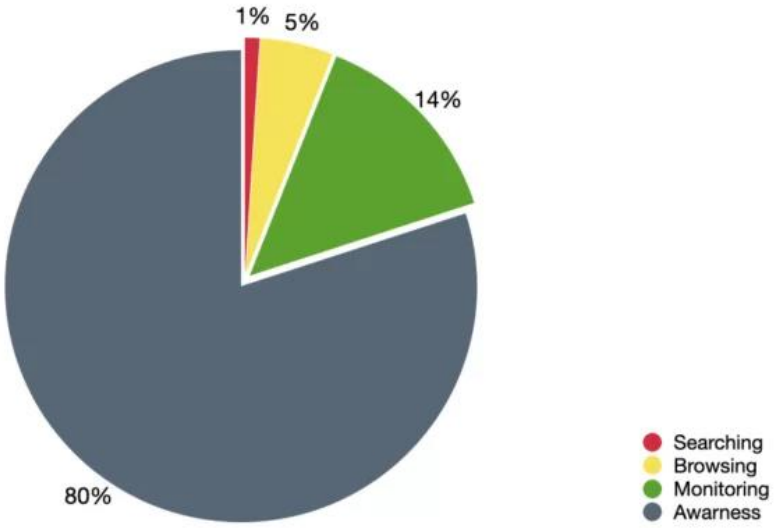
Breve analisi dei requisiti

Destinatari

La seguente applicazione web è pensata per essere utilizzata da chiunque intenda 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 utilizzare il servizio offerto, conosce già cosa vuole cercare e ha un livello di motivazione medio, caratterizzato da curiosità ma soprattutto 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 tale che le diverse tipologie di utenti siano in grado di trovare 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 IlTuoMeteo, quando si ricerca una città per vederne il clima per un eventuale viaggio programmato nei giorni immediatamente successivi.
- **Directed** – **Passive** → monitoring, nel caso di IlTuoMeteo, 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, IlTuoMeteo 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 le seguenti interfacce:

Index → Una pagina di presentazione del progetto, con uno stile semplice ma accattivante, troviamo un bottone per accedere al servizio meteo e un collegamento diretto ai profili GitHub nel footer della pagina. Inoltre presenta alcune animazioni per attirare l'attenzione dell'utente sugli elementi principali che la costituiscono, come il titolo e il bottone.

Meteo → La pagina vera e propria, 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. Inserendo un input viene restituito un output, questo procedimento è alla base della 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 effettuano richieste con metodo 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 da parte dell'utente 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, semplice 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, infatti, vengono aggiornati solamente i dati necessari. Infine, grazie allo stesso approccio, è stato possibile garantire un ciclo di vita indipendente ai dati, separando logicamente struttura, stile e contenuto.

Flusso dei dati

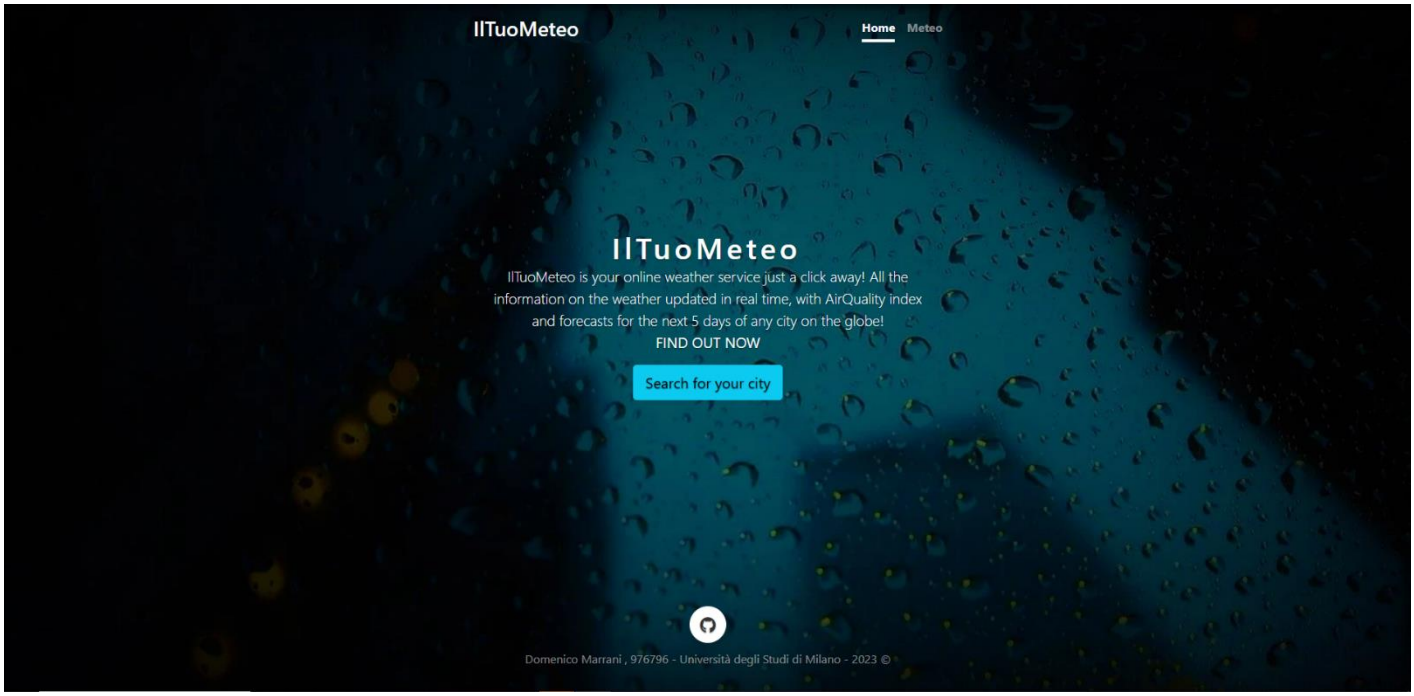
Il contenuto deve essere percepito come d'interesse per i destinatari ed è stato quindi espresso in uno stile adeguato, richiedendo il minimo sforzo da parte del destinatario e garantendo il massimo dell'informazione ricercata da quest'ultimo. I dati sono elementi indipendenti dalla struttura della pagina 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 tramite 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

Quest’applicazione non è a scopo di lucro, essa può essere considerata un esempio pratico del fatto che per inserirsi nel mondo dell’web e dell’economia dell’informazione non sia necessario un ingente patrimonio. IlTuoMeteo 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.

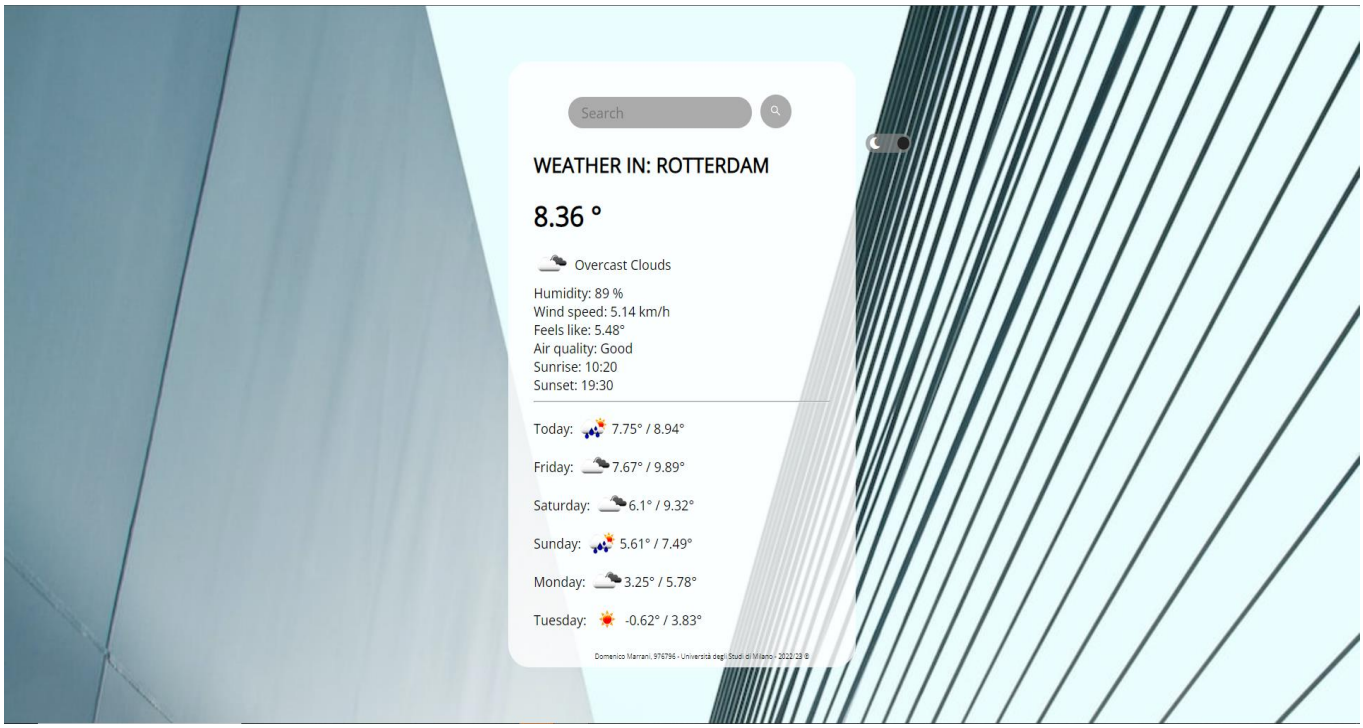
Panoramica Interfacce Utente

Index.ejs

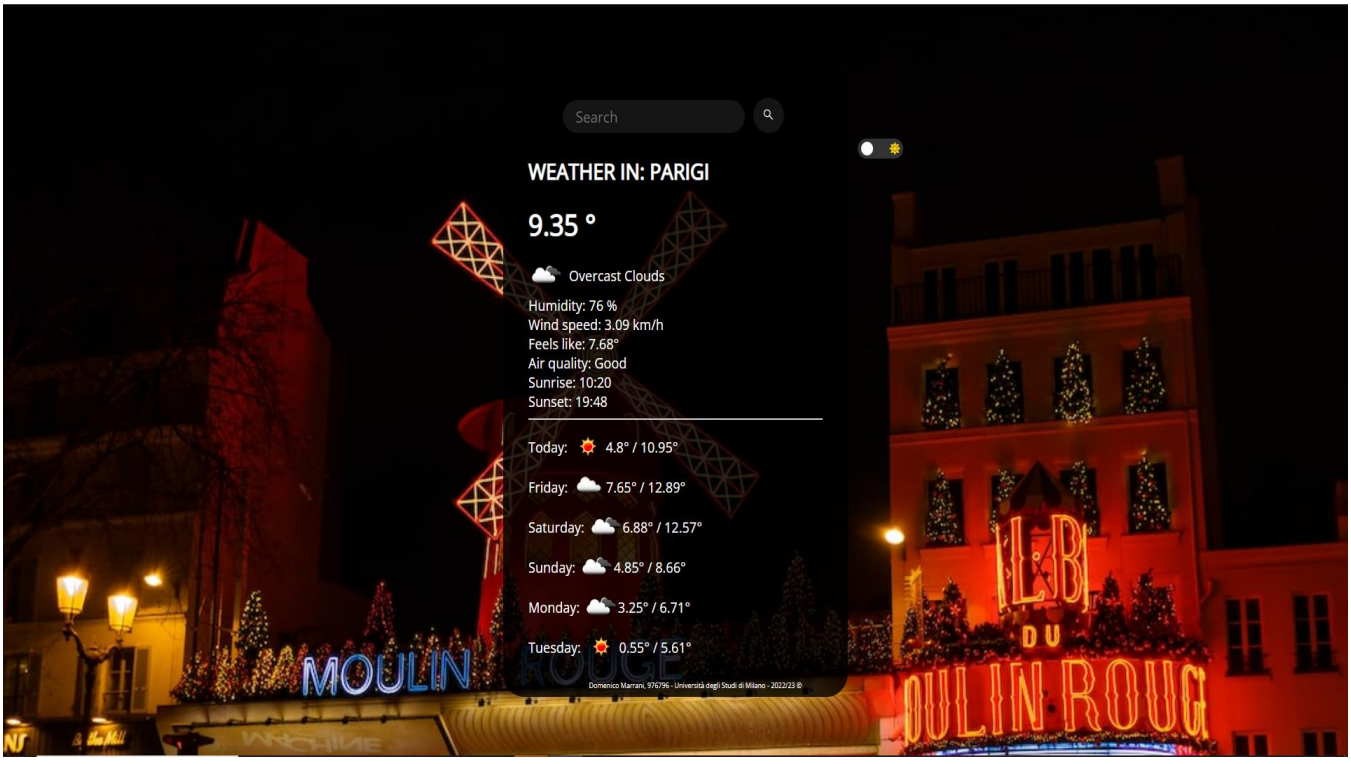


Meteo.ejs

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 per esempio 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 stati rimossi i segmenti di codice non utilizzati e superflui.

JavaScript

Il cuore dell'applicazione risiede in javascript, utilizzato sia lato client, ad esempio per gli script che permettono 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-OfService), 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 aumentando l'usabilità di quest'ultima.

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

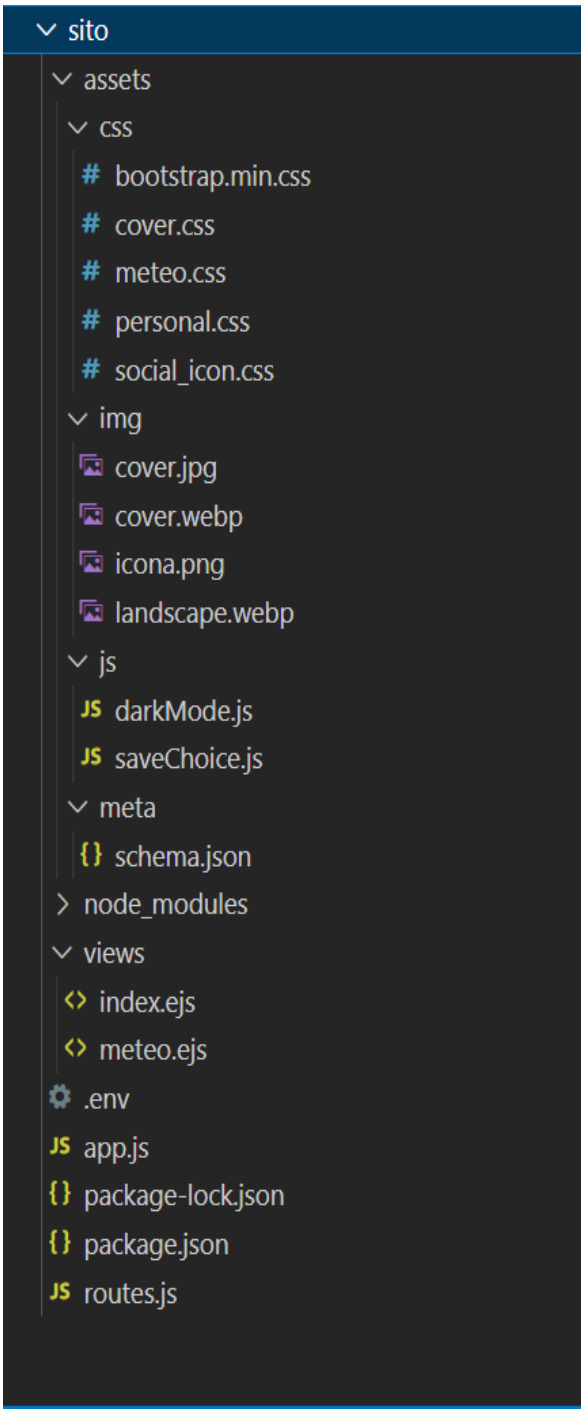
Il progetto sfrutta un semplice linguaggio di templating chiamato EJS (Embedded Javascript templates).
Questo linguaggio ha permesso di creare pagine dinamiche che assumono aspetti diversi in base ai dati ricevuti senza dover andare a modificare nulla.
é stato indispensabile utilizzare questa tecnologia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

Responsive Design

Tutte le pagine hanno un design responsivo, ottenuto tramite librerie (come Bootstrap) e CSS.
Questo permette di visualizzare su qualsiasi dispositivo la pagina, aumentando l'usabilità e la versatilità dell'applicazione.

Architettura delle risorse

Nella cartella assets troviamo delle sub-directories contenenti i fogli di stile, gli script javascript, le immagini per il sito web e uno schema JSON contenente i metadati nel caso in cui serva 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 dall'utente.

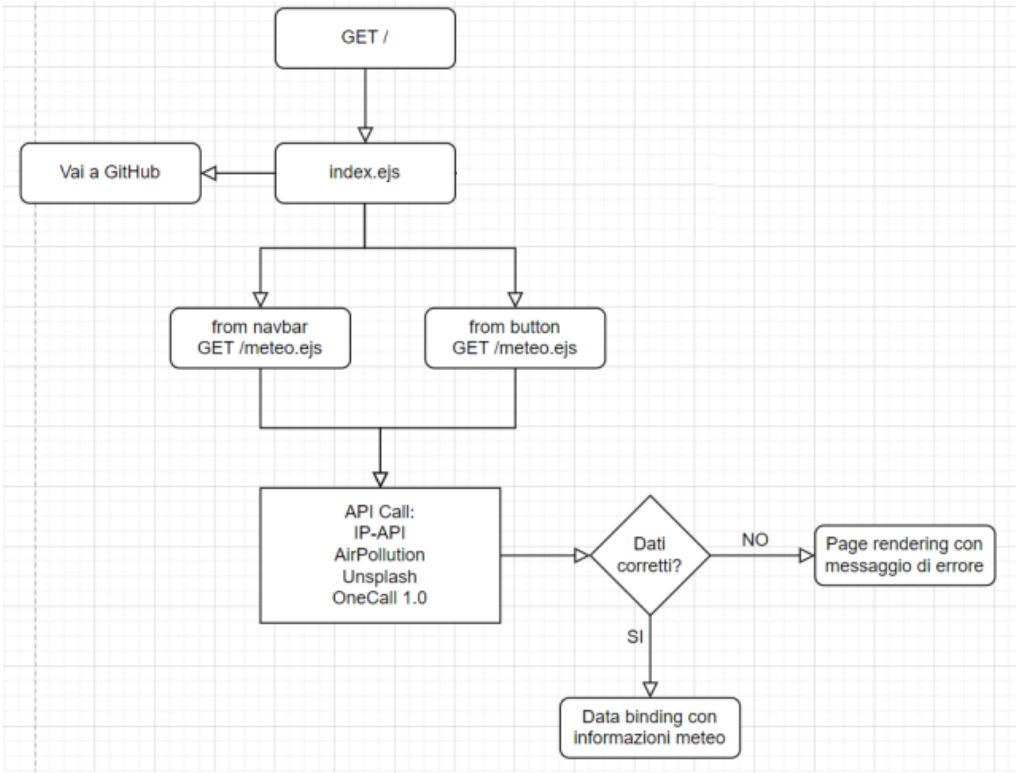
Nel file .env si trovano tutte le variabili di ambiente, come le API key e il numero di porta del server.

Package.json e package-lock.json contengono tutte le dependecies di node.js, cioè tutti i moduli necessari per il funzionamento.

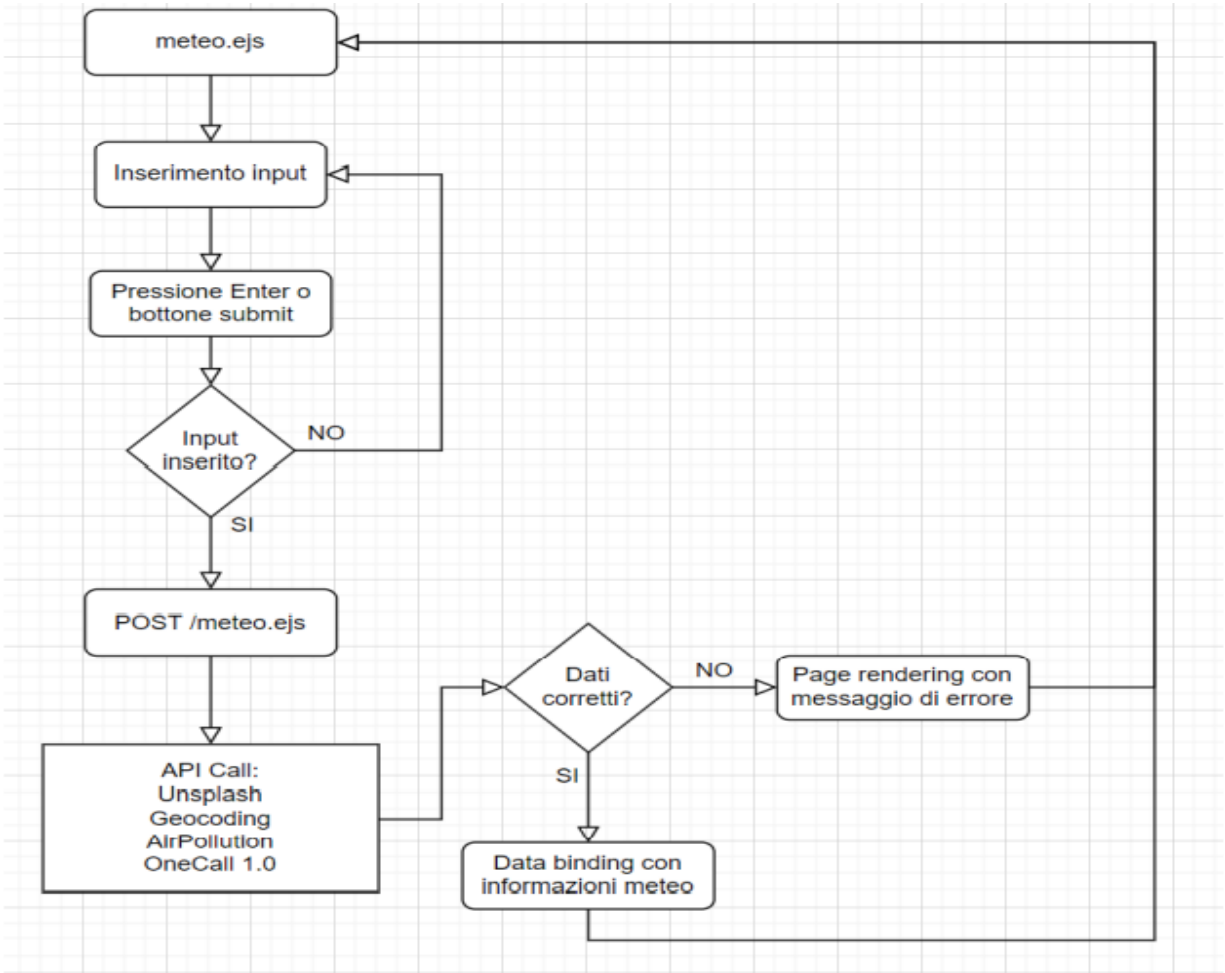
La cartella node_modules contiene le installazioni di tutti i pacchetti richiesti.

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 di Index.ejs



Funzionamento di Meteo.ejs



Sezioni del codice (IDE utilizzato: vs studio code)

Index.ejs

```
<div class="cover-container d-flex w-100 h-100 p-3 mx-auto flex-column">
  <header class="mb-auto">
    <div>
      <h3 class="float-md-start mb-0">IlTuoMeteo</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">IlTuoMeteo</h1>
    <p class="lead">IlTuoMeteo is your online weather service just a click away! All the information on the weather updated
      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/marranid" class="icon github" target="_blank">
        <div class="tooltip">Github</div>
        <span><i class="fab fa-github"></i></span>
      </a>
    </div>
    <p>Domenico Marrani , 976796  - Università degli Studi di Milano - 2023 &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" ><%= weatherin %>: <%= 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>
81            <img src="<%= imgtom %>" class="icon" loading="lazy" alt="icona meteo">
82            <div><%= mintom %>° / <%= maxtom %>°</div>
83          </div>
84          <div class="flex">
85            <div><%= day2 %>: </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

```
1   const express = require('express')
2   const Ddos = require('ddos')
3   const app = express()
4   var ddos = new Ddos ({burst:10, limit:10})
5
6   //Import routes
7   const routes = require('./routes.js')
8
9   app.use(express.urlencoded({ extended: true }))
10  app.use(ddos.express)
11
12  //Use view engine
13  app.set('view engine', 'ejs')
14
15  //Middleware route
16  app.use('/', routes)
17  app.use(express.static('assets'))
18
19  const PORT = process.env.PORT || 5000
20
21  app.listen(PORT, () => {
22    console.log(`Server starting at ${PORT}`)
23  })
```

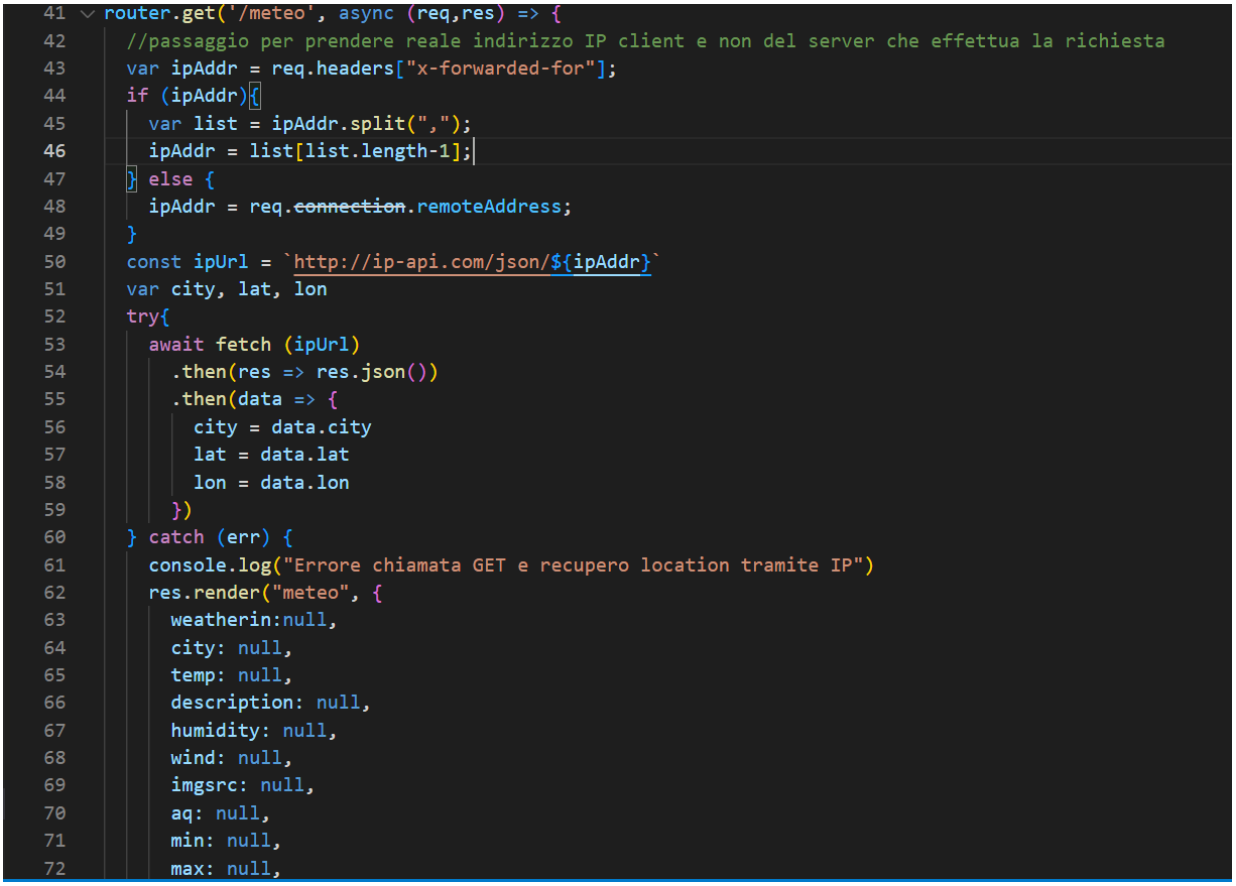
Deploying di un server express, con protezione anti-dos, funzioni di middleware e importing delle routes (gestite su file a parte)

Routes.js (1)-Dichiarazione di alcune funzioni per una maggiore modularità e manutenibilità

```
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
}
```

Route.ejs(2)- GET /Meteo.ejs con acquisizione ip per ricavare lat,long e city e infine il render della pagina.



```
41  router.get('/meteo', async (req,res) => {
42    //passaggio per prendere reale indirizzo IP client e non del server che effettua la richiesta
43    var ipAddr = req.headers["x-forwarded-for"];
44    if (ipAddr){
45      var list = ipAddr.split(",");
46      ipAddr = list[list.length-1];
47    } else {
48      ipAddr = req.connection.remoteAddress;
49    }
50    const ipUrl = `http://ip-api.com/json/${ipAddr}`
51    var city, lat, lon
52    try{
53      await fetch (ipUrl)
54        .then(res => res.json())
55        .then(data => {
56          city = data.city
57          lat = data.lat
58          lon = data.lon
59        })
60    } catch (err) {
61      console.log("Errore chiamata GET e recupero location tramite IP")
62      res.render("meteo", {
63        weatherin:null,
64        city: null,
65        temp: null,
66        description: null,
67        humidity: null,
68        wind: null,
69        imgsrc: null,
70        aq: null,
71        min: null,
72        max: null,
```

Routes.js (3) – POST /meteo.ejs con acquisizione e verifica dati e successivo rendering pagina in base all'esito.

```
390 router.post('/meteo', async (req,res) => {
391   const city = req.body.city
392   var backgroundLink = await getImage(city,process.env.UNSPLASH_KEY)
393   var lat,lon
394   const geocodingUrl = `http://api.openweathermap.org/geo/1.0/direct?q=${city}&limit=5&appid=${process.env.API_KEY}`
395   try{
396     await fetch(geocodingUrl)
397       .then(res => res.json())
398       .then(data => {
399         lat = data[0].lat;
400         lon = data[0].lon;
401       })
402     } catch (err) {
403       console.log("Errore nel Geocoding API Call")
404       res.render('meteo', {···
440       })
441     }
442     var aq = await getAQ(lat,lon,process.env.UNSPLASH_KEY)
443     if(lat || lon && aq){
444       //onecall 1.0 api
445       const weatherUrl = `https://api.openweathermap.org/data/2.5/onecall?lat=${lat}&lon=${lon}&units=metric&exclude=minute
446       try{
447         await fetch(weatherUrl)
448           .then(res => res.json())
449           .then(data => {
450             if(data.message === 'city not found' || data.message === 'wrong latitude' || data.message === 'wrong longitude'
451               res.render('meteo', {···
487               })
488             } else {
489               const index = ["Good", "Fair", "Moderate", "Poor", "Very poor"]
490               const days = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"]
491               var date = new Date ()
```

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

Index.ejs

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)

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)

Node.js

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.
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, 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 stato ottenuto grazie all'impiego del lazy loading per il caricamento delle immagini (nelle ultime versioni di Chrome aggiunto come attributo del tag img), dall'inserimento degli script javascript in fondo alla pagina, così da ritardarne il caricamento e ottenere da parte dell'utente l'impressione di 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.

Sicurezza

Sono state introdotte delle politiche di cybersecurity, come una protezione anti-ddos, 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.

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, IlTuoMeteo 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 come descritto nel Modello di valore.

https://github.com/marranid/IlTuoMeteo