Corso di Programmazione Web e Mobile

A.A. 2021-2022

Weather-4all

Luca Manfrin, 02049A

Autore: Luca Manfrin

Ultima modifica: 1 dicembre 2022

Prima modifica: 1 ottobre 2022

Indice

Luca Manfrin, 02049A

Weather-4all

Simple weather for everyone

Introduzione

Weather-4all è una WebApp inerente al meteo, la quale tramite apposito form, consente la ricerca di qualsiasi città nel mondo fornendo le informazioni inerenti al meteo del giorno corrente e fino ai cinque giorni successivi.

Tra le varie informazioni possiamo trovare: temperatura, umidità, velocità del vento, qualità dell'aria, orario dell'alba e del tramonto e infine di che giornata si tratta, ad esempio: nuvolosa, serena o molto altro.

L'obiettivo era quello di realizzare un sito ver la visione e ricerca del meteo di qualsiasi città tramite ricerca, la realizzazioone è stata completata grazie all'unione di diversi linguaggi di programmazione: HTML, CSS, JavaScript, JSON e di diverse tecnologie: API call, Local Storage, Node JS, Host.

Breve analisi dei requisiti

Destinatari

Weather-4all è stato progettato con lo scopo di poter essere utilizzato da tutti gli utenti, i quali per qualsiasi motivo nutrono il bisogno di conoscere il meteo di un determinata località tramite il minimo sforzo.

Il sito è realizzato completamente in inglese in modo tale da non essere esclusivamente per gli utenti italiani, ma per gli utenti provenienti da qualsiasi nazionalità o che abbiano una minima conoscenza dell'inglese. L'utente inoltre non dovrà nemmeno avere alcuna dote in ambito informatico per l'utilizzo del sito, ma semplicemente la curiosità del meteo di una determinata città.

Una volta caricato il sito verrà visualizzata una pagina di presentazione nella quale si può trovare una veloce descrizione del sito stesso e un bottone, il quale permetterà di accedere alla pagina dedicata alla ricerca e visualizzazione del meteo della città desiderata.

Grazie ad una progettazione di tipo responsive sarà possibile accedere al sito e visualizzarlo in modo ottimale sia da pc, tablet che da smartphone.

La realizzazione è avvenuta tramite il modello Marcia Bates, il quale divide le strategie di ricerca in quattro tipologie principali:

Diretta, attiva → searching: nel nostro caso quando l'utente ricerca una determinata città tramite l'apposita barra di ricerca

Diretta, passiva → monitoring: nel nostro caso quando l'utente visita la pagina del meteo solo per controllare le informazioni della località precedentemente cercataIn

Indiretta, attiva → browsing: nel nostro caso l'utente stava cercando un sito web, il quale permettesse di cercare il meteo ed è approdato su Weather-4all

Indiretta, passiva → being aware: nel nostro caso l'utente è venuto a conoscenza del sito grazie ad una inserzione pubblicitaria o tramite semplice passaparola da parte di un conoscente.

Modello di valore

La WebApp è stata realizzata senza scopo di lucro, in quanto si tratta di un progetto univeristario, la sua stessa realizzazione non ha previsto nessun tipo di costo in quanto sono stati usati solo ed esclusivamente servizi gratuiti, sia per le API call che per il servizio di hosting del sito.

Ciò non esclude la possibilità di monetizzare la WebApp, infatti con la semplice aggiunta di inserzioni pubblicitarie fornite ad esempio da GoogleAds è possible guadagnare in base al flusso di utenti che visiteranno il sito e cliccheranno sulle pubblicità presenti.

Flusso di dati

Il contenuto, il quale deve essere percepito come d'interesse per il destinatario è stato 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, il quale viene recuperato dal server attraverso una funzione di express che permette di accettare i dati di una richiesta POST accedendo al body della richiesta.

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.

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, il sito è stato pensato per essere accessibile a qualsiasi tipo di utente, da qualsiasi dispositivo. Utilizzando un approccio AJAX (Asynchronous JavaScript and XML) tramite 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 dei dati, separando logicamente struttura, stile e contenuto.

Interfacce

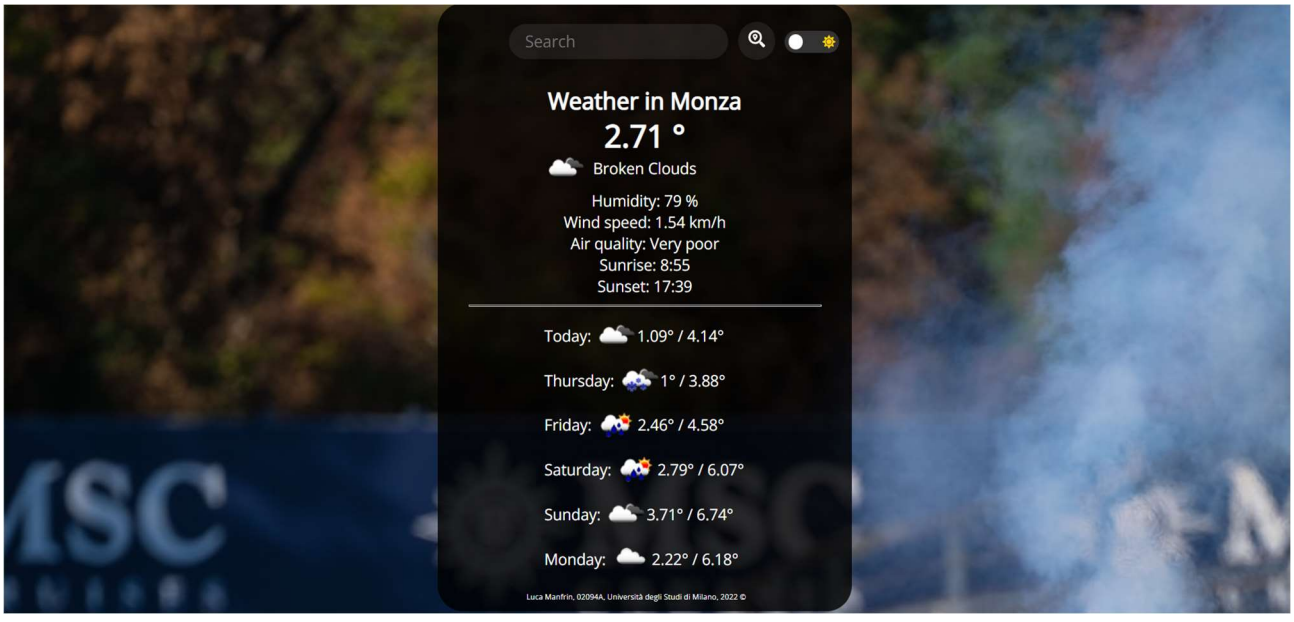
La WebApp è composta da due interfacce:

1: Index.ejs: è la landing page del sito, presenta uno sfondo scuro per riposare la vista, al centro possiamo trovare una breve descrizione del sito con un bottone di colore acceso in modo da catturare l'attenzione dell'utente, il cui nome fa intuire l'utilizzo, infatti serve per poter accedere alla seconda pagina nella quale si potrà cercare la città desiderata, infine nel fondo della pagina tramite tre bottoni che fungono da link si potrà accedere ai miei account GitHub, Linkedin e Instagram con sotto di essi delle mie informazioni.

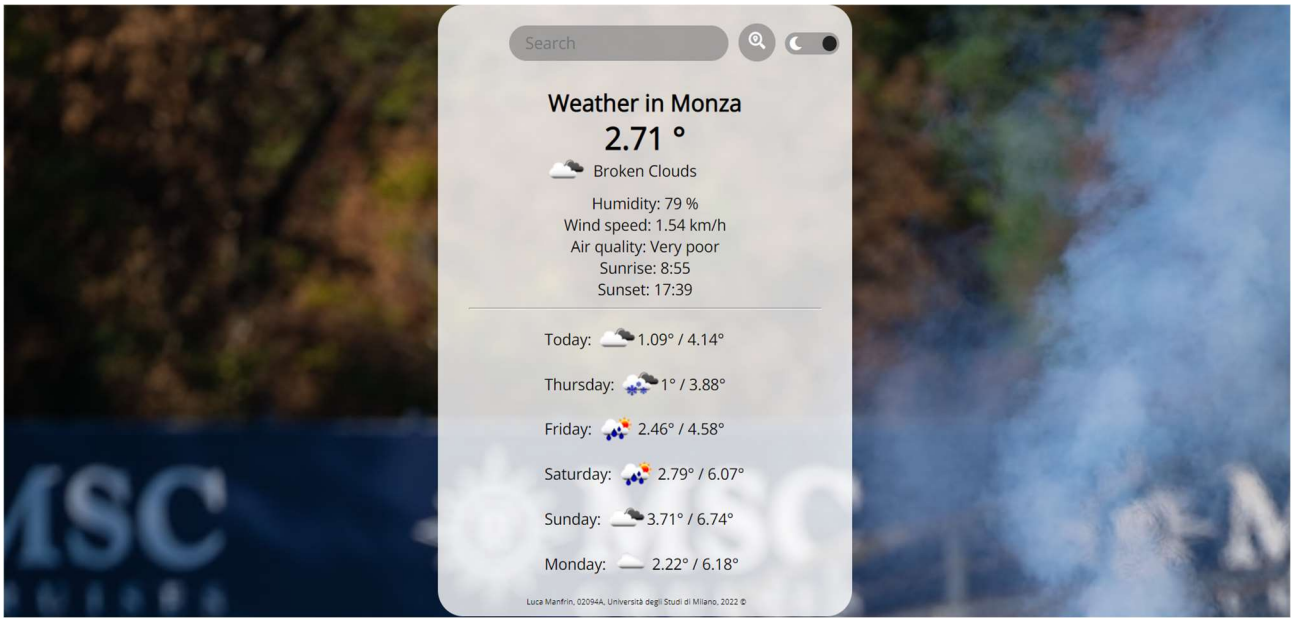
Index.ejs

2: Weather.ejs: è la pagina nella quale risiede il vero e proprio servizio di previsione metereologica, è stata utilizzata una gafica a modi card con possibilità di attivare e disattivare la DarkMode.
Tramite la barra di ricerca dopo aver inserito la località da cercare e aver premuto il tasto Enter o la lente di ingrandimento sulla destra verranno visualizzate tutte le informazioni climatiche inerenti alla città desiderata del giorno corrente e dopo una linea divisoria verranno mostrate in modo ridotto le informazioni inerenti ai cinque giorni successivi

Weather.ejs in versione DarkMode



Weather.ejs in versione LightMode



Le seguenti interfacce sono state realizzate in HTML per quanto riguarda la struttura della pagina e in CSS3 per quanto riguarda la parte grafica di esse

Tecnologie utilizzate

HTML

Tramite il linguaggio di markup HTML5 è stato possibile creare le due strutture delle pagine, con i relativi collegamenti ipertestuali, ad esse sono stati importati dei file esterni, come i fogli di stile in CSS relativi alla parte grafica di quest'ultime. Infine, tutte le pagine sono state validate tramite il sito www.validator.w3.org

CSS

L'utilizzo di CSS ha reso possibile l'implementazione della parte grafica, sia per quanto riguarda i vari colori utilizzati che la disposizione delle scritte, dei button e della card stessa, tramite lo strumento UnCSS è stato eliminato il codice in eccesso.

JavaScript

Tramite l'ausilio di JavaScript è stato realizzato lo script relativo alla DarkMode e memorizzare la scelta nei LocalStorage, entrambi lato client e gli script lato server grazie a Node.JS

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.

API

API utilizzate:

- OneCall API 1.0 by OpenWeather che restituisce tutti i dati relativi al meteo del giorno corrente e dei 5 giorni successivi. I dati utilizzati sono la temperatura, velocità del vento, l'umidità, temperatura massima e minima, l'icona del clima, descrizione del cielo, l'orario dell'alba e del tramonto.

- Geocoding API by OpenWeather che permette, partendo 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 uno sfondo 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.

Node.js

Sono stati utilizzati diversi moduli di Node.JS:

• 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 → è un modulo che permette l'uso di "fetch" lato server, il quale, è 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.

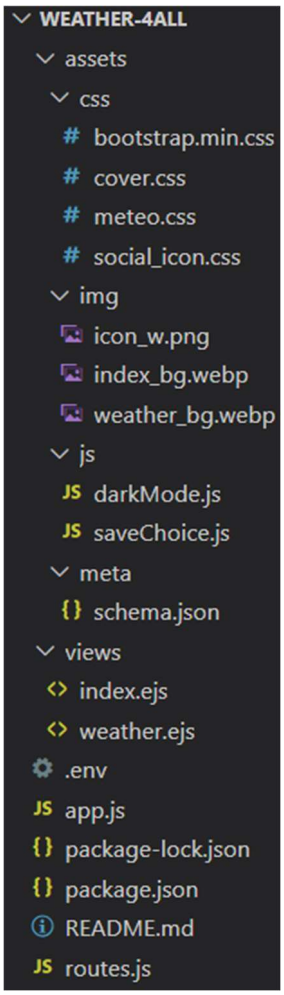
• Dotenv → si occupa di caricare tutte le variabili di ambiente contenute in un file .env dentro process.env, come le API key e la porta del server in caso non ce ne siano altre disponibili. Questo permette anche una maggiore segretezza di queste ultime

• 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

Utilizzati 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 volta che avviene il refresh della pagina.

Architettura



Il progetto è composto dalla cartella assets e views, la prima contiene a sua volta delle sotto-directory, ognuna dedicata ad un uso specifico, la prima riguarda la parte grafica del sito e contiene i fogli di stile cioè i file CSS, nella seconda sono contenute le immagini utilizzate come sfondo, nella terza i due script javascript relativi alla DarkMode e ai LocalStorage e infine nell'ultima si trova uno schema JSON contenente i metadati per l'indicizzazione della pagina dai motori di ricerca (creato seguendo lo schema "WebPage" presente su schema.org).

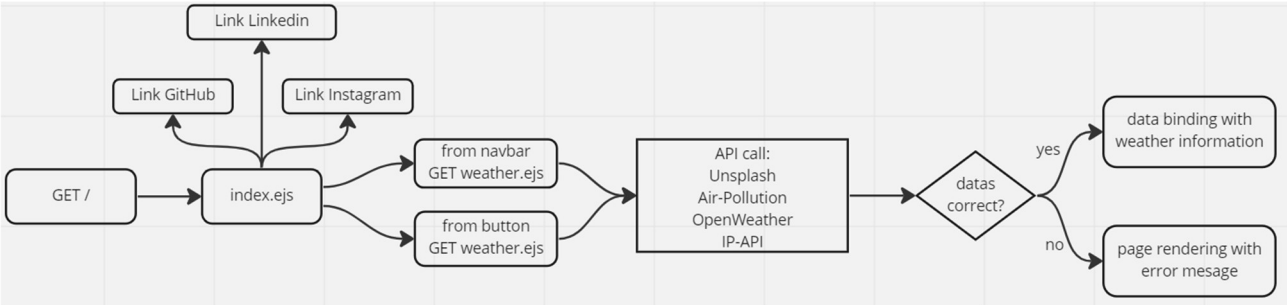
Nella seconda si trovano le due interfacce utente, index.ejs e weathr.ejs, in quest'ultima arriverà il data binding con le informazioni della città richiesta. Package.json e package-lock.json contengono tutte le dependecies di node.js, cioè tutti i moduli necessari per il funzionamento

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

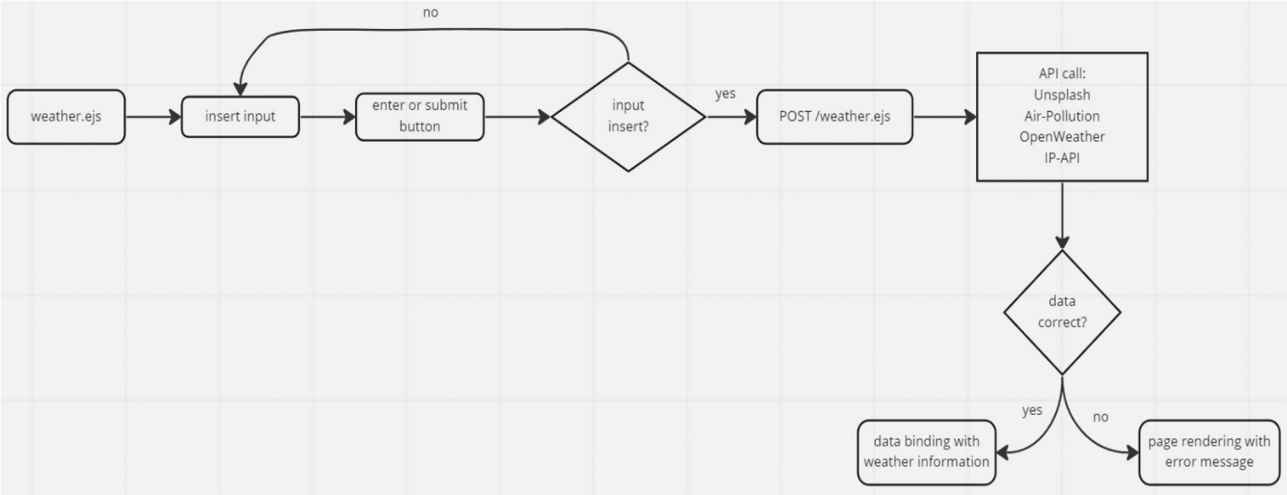
Nel file app.js si trova la configurazione di un server express, con relative funzioni di middleware, mentre nel file routes.js si trova la gestione del routing e quindi degli endpoint raggiungibili dall'utente, con il rendering delle views e l'impostazione delle variabili EJS.

Descrizione delle risorse

Funzionamento di Index.ejs



Funzionamento di weather.ejs



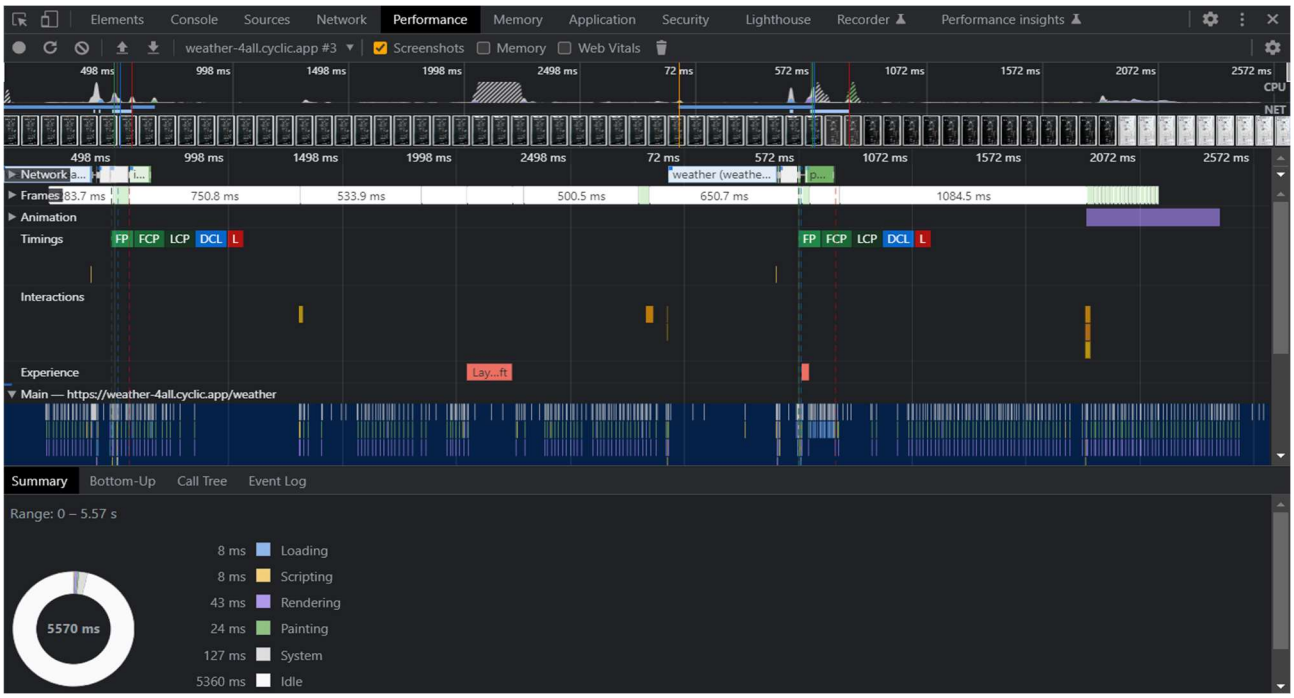
Responsive

La realizzazione del progetto è stata fatta in maniera tale da avere un design di tipo resposive, in modo tale da rendere la lettura piacevole per qualsiasi tipo di dispostivo utilizzato per la visione del sito, questo è stato reso possibile grazie all'utilizzo di librerie, ad esempio, Bootstrap o tramite CSS.

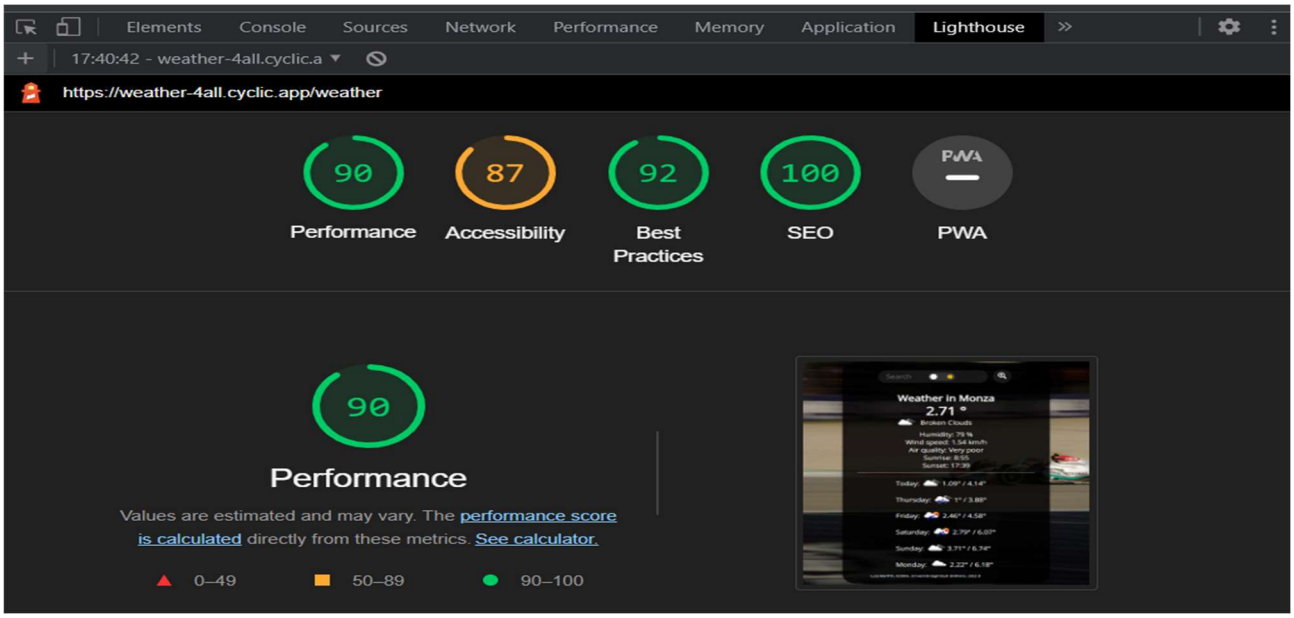
Di seguito lo screenshot eseguito da pc portatile e da telefono per dimostrarne la dinamicità:

Prestazioni

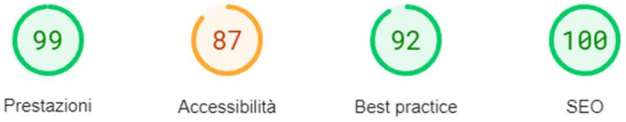
Panoramica delle prestazioni, come si può notare è molto veloce grazie a progressivi miglioramenti durante lo sviluppo, ad esempio: l'introduzione del lazy loading per il caricamento delle immagini, il quale, viene aggiunto come attributo nel tag dell'immagine, l'inserimento degli script JavaScript alla fine della pagina, in modo tale da ritardarne il caricamento e mostrare all'utente un caricamento più veloce, l'utilizzo dello strumento UnCSS che ha permesso l'eliminazione del codice CSS in eccesso, rendendo il file più corto e di conseguenza più leggero, l'utilizzo di uno sfondo in formato .webp a discapito di .jpg il quale è più pesante e più lento, la gestione lato server ha permesso un guadagno di prestazioni elevato anche grazie all'utilizzo delle API call, sfruttando un approccio non-blocking, reso posssibile anche grazie al motore V8 di node.js che garantisce elevate prestazioni, infine l'utilizzo del templating e data binding tramite EJS con approccio AJAX che ha quasi azzerato i tempi di rendering e painting in quanto la pagina è già stata disegnata durante la richiesta GET e successivamente con la richiesta POST vengono aggiornati i dati.

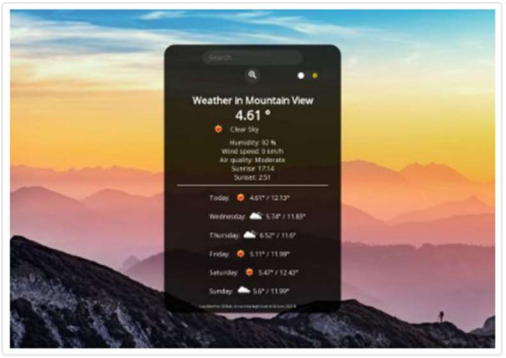


Di seguito i risultati ottenuti dal test effetuato dal lighthouse report di Google Chrome



Di seguito i risultati del report del sito PageSpeedInsight





Sicurezza

Sono state introdotte delle politiche di cybersecurity, come una protezione antidos, 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).

Deploy

L'applicazione, una volta finita, per essere disponibile non solo in maniera locale, tramite il servizio offerto da Cyclic.sh, il quale permette il deploy di qualsiasi app interamente tramite un infrastruttura cloud serverless è stata deployata semplicemente indicando il repository GitHub del mio progetto. Inoltre, permette la creazione di un proprio dominio di secondo livello in questo formato: nome.cyclic.app, tra i vari servizi contenuti nel piano gratuito possiamo trovare un sistema di log in tempo reale, un grafico il quale mostra il numero di chimate API effettuate, una schermata con la data e il tempo di tutti i deploy effettuati e molte altre features

Codice

Index.ejs: pagina principale contenente navabar, testo, bottone e footer per le icone dei social

```
<header class="mb-auto">
  <div>
    <h3 class="float-md-start mb-0">Weather-4all</h3>
    <!-- Nav bar con collegamenti alle pagine -->
    <div float="right">
    <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="/weather">Weather</a>
    </div>
    </nav>
  </div>
</header>
<!-- Main della pagina, con titolo, descrizione e bottone -->
<main class="px-3">
  <h1 class="typewriter" color="#0dcaf0">Weather-4all</h1><br><br>
  <p class="lead">Weather-4all it's your online weather service simple and for everybody! All the information on the weather updated in real t
    for the next 5 days <br>of any city on the globe!<br></p>      You, 3 settimane fa • modifiche …
  <p class="lead">
    <a href="/weather" class="btn btn-lg btn-info animate__animated animate__jackInTheBox">Search a 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/LucaMan32" 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/luca-manfrin-08a070181/" class="icon linkedin" target="_blank">
      <div class="tooltip">Linkedin</div>
      <span><i class="fab fa-linkedin"></i></span>
    </a>
    <a href="https://www.instagram.com/lucamanfrin__/" class="icon instagram" target="_blank">
      <div class="tooltip">Instagram</div>
      <span><i class="fab fa-instagram"></i></span>
    </a>
```

Weather.ejs: codice contenente parte del data binding con EJS e le relative condizioni, di sotto si può vedere la prima parte della card

```
    <div class="weather">  <!-- creazione div relativo al meteo con div per ogni campo -->
      <% if(city !== null) { %>
        <h2 class="city">Weather in <%= city %></h2>
      <% } %>
      <% if(temp !== null) { %>
        <h1 class="temp"><%= temp %> °</h1>
      <% } %>
      <div class="flex">
        <% if(imgsrc !== null) { %>
          <img src="<%= imgsrc %>" alt="icona meteo" class="icon" loadin="lazy">
        <% } %>
        <% if(description !== null) { %>
          <div class="description"><%= description %></div>
        <% } %>
      </div>
      <% if(humidity !== null) { %>
        <div class="humidity">Humidity: <%= humidity %> %</div>
      <% } %>
      <% if(wind !== null) { %>
        <div class="wind">Wind speed: <%= wind %> km/h</div>
      <% } %>
      <% if(aq !== null) { %>
        <div class="aq">Air quality: <%= aq %></div>
      <% } %>
      <% if(sunrise !== null) { %>
        <div class="sunrise">Sunrise: <%= sunrise %></div>
      <% } %>
      <% if(sunset !== null) { %>
        <div class="sunset">Sunset: <%= sunset %></div>
      <% } %>
    </div>
```

App.js: deploying del server express, protezione anti-dos, funzioni di middleware e importing delle routes

```
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}`)
})
```

Router.js: dichiarazione di alcune funzioni inerenti all'acquisizione delle API

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

async function getImage (city,key) { // funzione per l'acquisizione dello sfondo
  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("Error with the loading of the background...")
    backgroundLink = "/img/weather_bg.webp"  // in caso di errore viene caricato uno sfondo predefinito
  }
  return backgroundLink
}

async function getAQ (lat,lon,key) {  // funzione per l'acquisizione della qualità dell'aria
  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("Error in the fetch API call for the Air Quality Index.")  // messaggio in caso di errore
  }
  if(isNaN(aq))
    aq = "Error!"
  return aq
}
```

POST /meteo.ejs con rendering pagina

```
158             res.render('weather', {
159               city: city,
160               temp: data.current.temp,
161               description: data.current.weather[0].description,
162               humidity: data.current.humidity,
163               wind: data.current.wind_speed,
164               imgsrc: "https://openweathermap.org/img/w/" + data.current.weather[0].icon + ".png",
165               aq: index[aq-1],
166               min: data.daily[0].temp.min,
167               max: data.daily[0].temp.max,
168               imgtoday: "https://openweathermap.org/img/w/" + data.daily[0].weather[0].icon + ".png",
169               imgtom: "https://openweathermap.org/img/w/" + data.daily[1].weather[0].icon + ".png",
170               mintom: data.daily[1].temp.min,
171               maxtom: data.daily[1].temp.max,
172               imgdayafter: "https://openweathermap.org/img/w/" + data.daily[2].weather[0].icon + ".png",
173               mindayafter: data.daily[2].temp.min,
174               maxdayafter: data.daily[2].temp.max,
175               unsplash: backgroundLink,
176               day1: days[(date.getDay() + 1) % 7],
177               day2: days[(date.getDay() + 2) % 7],
178               day3: days[(date.getDay() + 3) % 7],
179               imgday3: "https://openweathermap.org/img/w/" + data.daily[3].weather[0].icon + ".png",
180               min3: data.daily[3].temp.min,
181               max3: data.daily[3].temp.max,
182               feels: data.current.feels_like,
183               day4: days[(date.getDay() + 4) % 7],
184               imgday4: "https://openweathermap.org/img/w/" + data.daily[4].weather[0].icon + ".png",
185               min4: data.daily[4].temp.min,
186               max4: data.daily[4].temp.max,
187               day5:  days[(date.getDay() + 5) % 7],
188               imgday5: "https://openweathermap.org/img/w/" + data.daily[5].weather[0].icon + ".png",
189               min5: data.daily[5].temp.min,
190               max5: data.daily[5].temp.max,
191               sunrise: (sunrise.getHours()+2) + ":" + sunrmin,
192               sunset: (sunset.getHours()+2) + ":" + sunsmin
193             })       Luca Manfrin, mese scorso • Add files via upload
```

DarkMode.js: implementazione delle funzioni relative alla DarkMode e LightMode

```
1   function setLight () {
2       document.getElementById('card').style.backgroundColor = "rgba(255, 255, 255, 0.8)"
3       document.getElementById('card').style.color = "black"
4       document.getElementById('search-bar').style.backgroundColor = "rgba(135, 135, 135, 0.7)"
5       document.getElementById('search').style.backgroundColor = "rgba(135, 135, 135, 0.7)"
6   }
7
8   function setDark () {
9       document.getElementById('card').style.backgroundColor = "#000000d0"
10      document.getElementById('card').style.color = "white"
11      document.getElementById('search-bar').style.backgroundColor = "#7c7c7c2b"
12      document.getElementById('search').style.backgroundColor = "#7c7c7c2b"
13  }
14
15  document.addEventListener('DOMContentLoaded', function (){
16      var checkbox = document.getElementById('switch');
17
18      checkbox.addEventListener('change', function () {
19          if(checkbox.checked) {
20              setLight()
21          } else {
22              setDark()
23          }
24      });
25  });
```

SaveChoice.js: implementazione dei Local Storage per il salvataggio della scelta

```
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
  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 = {
```

Conclusioni

La realizzazione del progetto è iniziata durante le ore di laboratorio e successivamente è stata sviluppata e conclusa fuori dall'ambiente universitario. L'obiettivo futuro è di migliorare il progetto con l'aggiunta di diverse feature, prima di tutte una schermata di login nella quale sarà possibile effettuare il login o eventualmente registrarsi per i nuovi utenti, la quale accederà all'attuale landing page la quale permetterà oltre che a raggiungere la pagina dedicata al meteo anche un area apposita nella quale sarà possibile visualizzare graficamente tutte le informazioni delle varie ricerche effettuate con relativo salvataggio il quale sarà possibile confrontare in maniera offline su un app mobile dedicata per un accesso più rapido. Infine, per finanziare il progetto può essere valutata l'opzione dell'inserimento di pubblicità in maniera non invasiva le quali tramite i ricavi permetteranno l'acquisto di servizi aggiuntivi delle varie API e servizio di hosting per poter migliorare ulteriormente il progetto e fornire maggiori informazioni sul meteo.

Nota bibliografica e sitografica

https://pagespeed.web.dev

https://www.cyclic.sh/

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://ejs.co/

https://www.youtube.com