Corso di Programmazione Web e Mobile

A.A. 2021/2022

Gabriele Stentella
984491
Prima modifica: 21 maggio 2022
Ultima modifica: 13 giugno 2022

Corso di Programmazione Web e Mobile

1. Introduzione

Il progetto presentato è stato realizzato nel percorso di laboratorio, si tratta di un'applicazione web che fornisce dati meteo di alcune città importanti nel mondo, oltre a costruire dinamicamente la pagina con le condizioni meteorologiche di qualsiasi località cercata tramite la barra di ricerca. L'applicazione ha lo scopo di offrire grafica responsiva e minimale, sviluppata con il framework bootstrap, in modo da adattarsi a un target di utenti interessati a un design molto semplice. È stato sviluppato un server locale, utilizzando il framework NodeJS, al fine di permettere di richiamare l'applicazione tramite protocollo http e alcune funzionalità introdotte (la segnalazione di bug e l'invio di consigli, l'autenticazione) utilizzano un database locale gestito con MongoDB. Altre tecnologie esterne utilizzate comprendono i servizi API di openweathermap.org e unsplash.com, rispettivamente per fornire i dati meteorologici aggiornati e le fotografie.

2. Analisi dei requisiti

Destinatari

L'applicazione realizzata è destinata a utenti che, secondo il modello di Marcia Bates, possono essere definiti attivi-consapevoli, grazie alla barra di ricerca essi possono cercare esattamente la città di cui vogliono sapere le informazioni, inoltre nel processo di registrazione è richiesto di indicare il nome della città preferita, quindi anche in questo caso è richiesto un grado alto di consapevolezza; viene lasciato anche un certo spazio a utenti attivi non consapevoli, infatti nella home page si trovano alcune città importanti del mondo alle cui pagine meteo l'utente può accedere semplicemente con un click, senza bisogno di cercare, in questo caso non è quindi richiesta una consapevolezza, perchè è la stessa applicazione che propone i contenuti. Il

target è quello di persone che viaggiano molto per lavoro, infatti le città inserite nella home sono i luoghi principali del business, inoltre tramite la registrazione o l'autenticazione è possibile vedere previsioni accurate per le successive 5 ore, proprio perchè è pensato per persone che devono trascorrere periodi molto brevi in una città. È stata posta una particolare attenzione al design, l'applicazione è infatti destinata a persone che amano il minimalismo e che sono predisposte a fare un piccolo sforzo di apprendimento iniziale per comprendere l'interfaccia spoglia, pur di avere un design immediato che gli fornisca molte informazioni in modo compatto.

Proprio perchè orientata a un target di lavoratori, l'applicazione è pensata per essere ben fruibile da ogni dispositivo, dal laptop, al cellulare, al tablet, grazie all'interfaccia responsiva e ai piccoli accorgimenti che consentono di mantenere un design minimale anche su schermi piccoli, nascondendo elementi non essenziali quando si utilizzano tali display.

Modello di valore

L'applicazione vede il suo valore nell'elaborazione e aggregazione dei dati forniti dalle API, oltre alla presentazione di tali dati in un'interfaccia semplice, per esempio sono state integrate le icone di bootstrap in sostituzione di quelle fornite da openweather; gli utenti possono essere spinti ad usare l'app attratti sia dal design, sia dalle funzionalità utili specificatamente per il target a cui l'app è indirizzata. Al fine di mantenere una delle sue caratteristiche peculiari, ovvero il design minimale, l'applicazione non è pensata per ospitare advertising, dunque il modello di business consiste nell'offrire il servizio gratuitamente, in modo da ottenere una buona base di utenti disposta a pagare per funzionalità premium, come può essere la possibilità di inserire il proprio piano di viaggio per avere le previsioni orarie di ogni città in cui si prevede di fermarsi; la base di utenti attiva, essendo composta da persone che viaggiano molto per lavoro, consentirebbe anche di ottenere un bacino di utenti internazionale,

sfruttando il meccanismo del passaparola che si creerebbe fornendo agli utenti il servizio che desiderano, a tal fine si renderebbe necessaria una traduzione del sito web in inglese.

Flusso dei contenuti

I dati dell'applicazione sono reperiti tramite API e dunque in continuo aggiornamento da parte dei gestori delle API, questo fa si che l'applicazione, una volta avviata, non abbia bisogno di particolari attività di rinnovamento, dunque si possono concentrare le attività per ampliarla inserendo nuove funzionalità.

I contenuti principali sono i dati meteo e le fotografie:

1. **I dati meteo** sono reperiti tramite chiamate API al servizio openweathermap.org, sono state utilizzate due API distinte, infatti nella homepage vengono mostrate solo le condizioni meteorologiche attuali, mentre nella pagina costruita dinamicamente a seguito della ricerca, così come nella pagina privata, vengono mostrate anche le previsioni; è da notare che al fine di offrire un design minimale le icone che vengono normalmente restituite dalla API sono state sostituite con le più pulite icone di bootstrap, effettuando il download in locale delle icone appropriate, rinominandole opportunamente in modo che i nomi coincidessero con i codici forniti dalla API e infine costruendo il percorso file utilizzando proprio tale codice.
   Per ottimizzare il caricamento di queste risorse è stato implementato un webworker che si occupa di eseguire le chiamate API e restituire allo script principale i dati una volta ricevuti.
2. **Le fotografie** possono essere distinte in due categorie, ovvero quelle statiche e dunque salvate in locale, e quelle che si adattano al contesto e quindi sono fornite tramite chiamate API al servizio unsplash.com. Nella home page si trovano le fotografie statiche per le città più importanti, ma nella sezione in cui viene fornito il meteo per il luogo dove si trova l'utente

viene utilizzato unsplash per fornire la foto del luogo o, in assenza di tale reperto, una foto trovata attraverso una ricerca effettuata inserendo la descrizione della condizione meteorologica fornita dalla API openweather. Nella pagina costruita dinamicamente con la ricerca vengono inserite le fotografie del luogo cercato solo se presenti in unsplash. Anche in questo caso, per ottimizzare il caricamento delle fotografie è stato implementato un webworker.

È da notare che le informazioni fornite dalle API sono utilizzabili liberamente anche per applicazioni a fini di lucro, non è dunque necessaria nessuna licenza.

Altri dati, quali i consigli degli utenti e i loro dati di log, sono conservati in un database locale gestito con MongoDB, nel primo caso solo al fine di rendere agevole la consultazione da parte dello sviluppatore, nel secondo per consentire l'autenticazione.

Aspetti tecnologici

Le pagine sono sviluppate in HTML5 e validate con w3c validator, i template per le due pagine che devono essere costruite dinamicamente in base alle richieste degli utenti sono sviluppate secondo il formato ejs; lo stile è implementato attraverso CSS, in parte scritto nei file style.css presenti nel progetto, in parte derivante dai template bootstrap, in particolare per gli aspetti responsivi delle pagine. Ogni pagina ha un suo file Javascript che serve per implementare il funzionamento sia di aspetti puramente grafici, come la dark mode, sia per le chiamate API e la comunicazione con il server.

Le chiamate API sono effettuate attraverso fetch, un modo più moderno e ottimizzato rispetto alle chiamate ajax, inoltre dopo ogni fetch viene utilizzato il metodo json() per ottenere i dati forniti dalla API in formato json.

Viene utilizzato Local Storage al fine di memorizzare la preferenza dell'utente per quanto riguarda la selezione della modalità di visualizzazione

(dark o light mode), in modo da poter caricare la pagina con la stessa modalità dell'ultima visita effettuata nelle ultime 24 ore.

Nel codice html della home page è implementato un oggetto json che, secondo lo standard schema.org, costituisce i metadati della pagina, fornendo gli elementi di cui il browser necessita per indicizzare al meglio la web app nei risultati di ricerca.

Il server locale, costruito grazie a NodeJS, fornisce le risorse per la navigazione nel sito e si occupa di costruire le pagine dinamicamente per quanto riguarda la ricerca e la pagina privata, effettuando le chiamate API attraverso axios, i dati sono poi gestiti in formato JSON. Il server si occupa anche di creare un cookie necessario per memorizzare la città scelta dall'utente in fase di registrazione, in modo da poter fornire la pagina personale costruita in base a tale città, la presenza del cookie è inoltre il fattore determinante che il browser del client utilizza per capire se caricare la pagina personale o la pagina di login, infatti il cookie viene creato solo dopo la fase di autenticazione, diventando, di fatto, un fattore identificativo per gli utenti che si sono registrati.

Anche se il cookie può essere considerato sicuro anche per memorizzare informazioni sensibili, si è preferito non utilizzarlo per i dati di login, i dati di autenticazione sono memorizzati esclusivamente nel database locale MongoDB, a ogni utente è assegnato un token, creato grazie al modulo Json Web Token a partire dallo username dell'utente, inoltre tale identificativo scade dopo 5h, il tempo per cui vengono date le previsioni nella pagina privata, poiché si presume che un utente si registri per vedere le previsioni per 5 ore, poi potrebbe voler cambiare la preferenza della città da visualizzare; la password è memorizzata dopo essere criptata attraverso il modulo bcryptjs che utilizza PBKDF (password based key derivation function) per creare l'hash corrispondente alla password, e fornisce anche un metodo

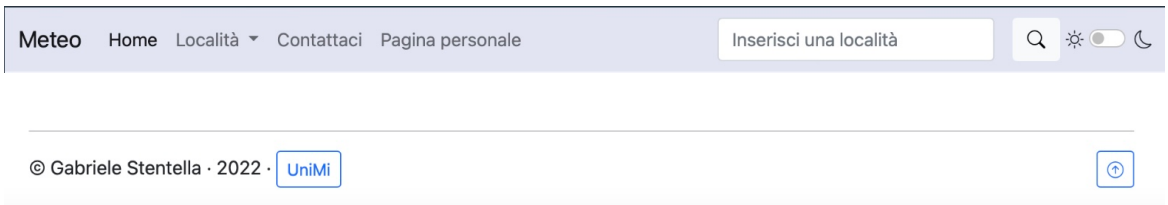
che compara la password memorizzata con quella inserita dall'utente senza esporre in chiaro l'informazione nel database.

Il routing e la creazione dinamica di pagine partendo da template sono gestiti con il framework express, il framework di riferimento per la gestione delle applicazioni lato server in NodeJS.

Oltre a gestire le richieste get per il rooting verso le pagine e le richieste post per il login e la registrazione, il server gestisce anche due richieste get aggiuntive, get_errors e cookies, che forniscono al client rispettivamente informazioni su eventuali errori, nello specifico errori di accesso, e il nome della città preferita dell'utente memorizzata nel cookie, se presente.

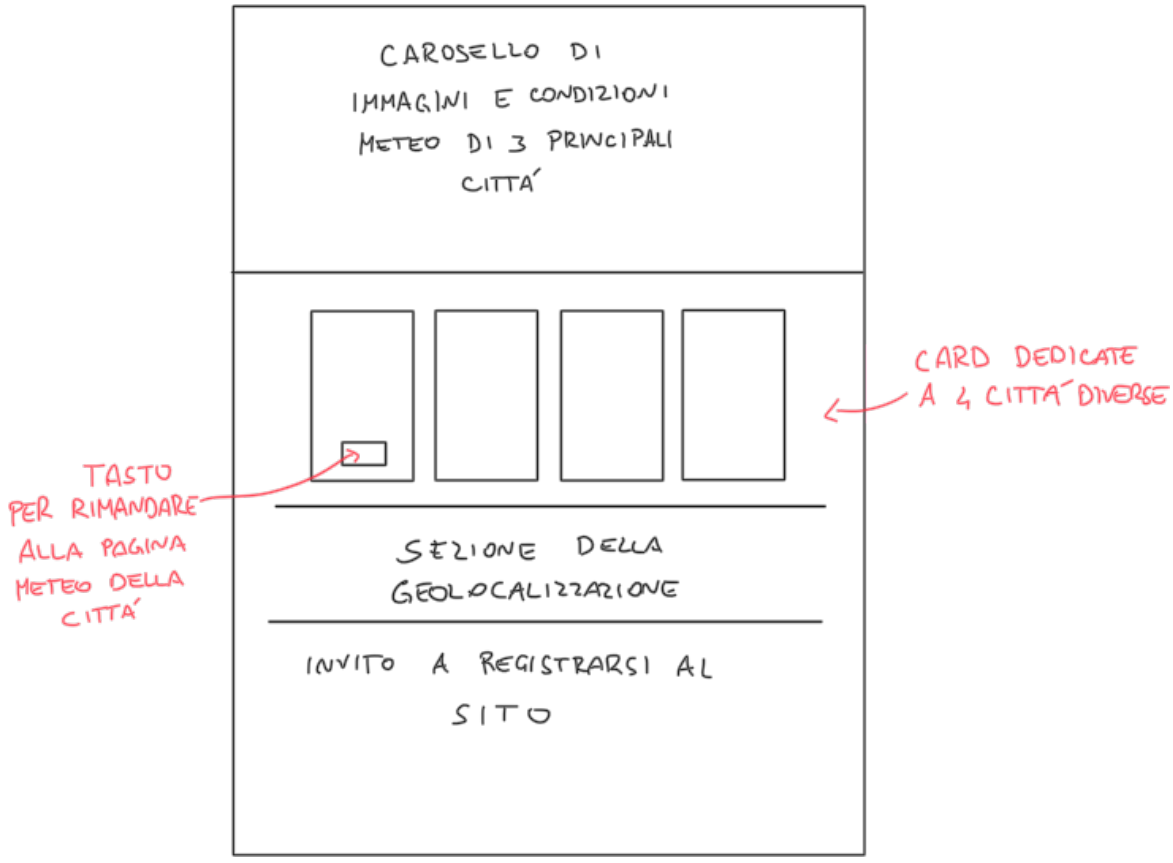
3. Interfacce

Gli elementi comuni a tutta l'applicazione sono la barra di navigazione e il footer: nella prima troviamo i tasti che portano alle pagine meteo di tre località principali, alla pagina di contatto e a quella personale, inoltre è presente una barra di ricerca e il tasto per attivare e disattivare la dark mode; nel footer è inserito il nome dell'autore e l'anno dell'ultima modifica al sito, oltre al link alla pagina web di UniMi e a un tasto per tornare in cima alla pagina.



Nella home page troviamo un carosello con le foto e i dati meteo attuali di tre città importanti, quattro cards che presentano foto e meteo di altrettante città oltre a un tasto che, se premuto, avvia la procedura per costruire la pagina meteo della città a cui la card si riferisce; sono inoltre presenti due sezioni, una contenente un invito a registrarsi al sito, l'altra

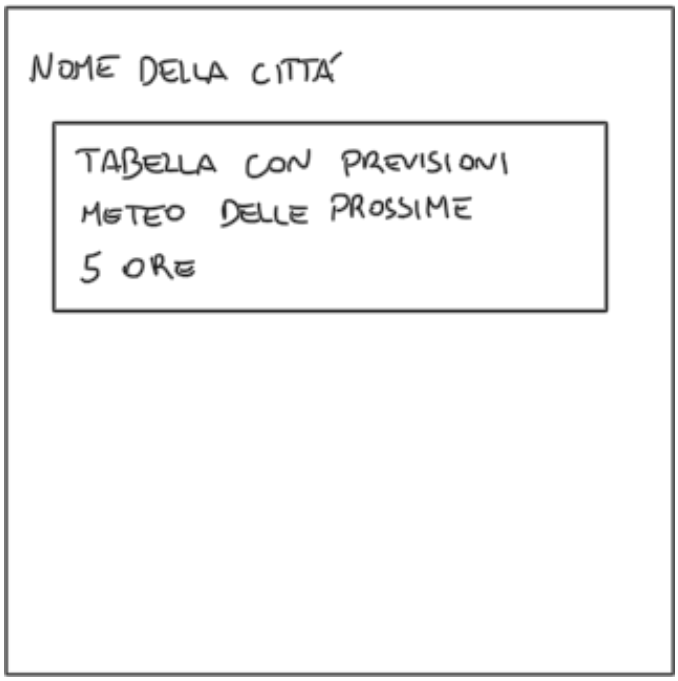
dedicata alle informazioni sul luogo in cui si trova l'utente. Tutte le immagini presenti sono salvate in locale (così come le icone del meteo) fatta eccezione per l'immagine nella sezione della geolocalizzazione che è invece fornita tramite una chiamata al servizio unsplash.



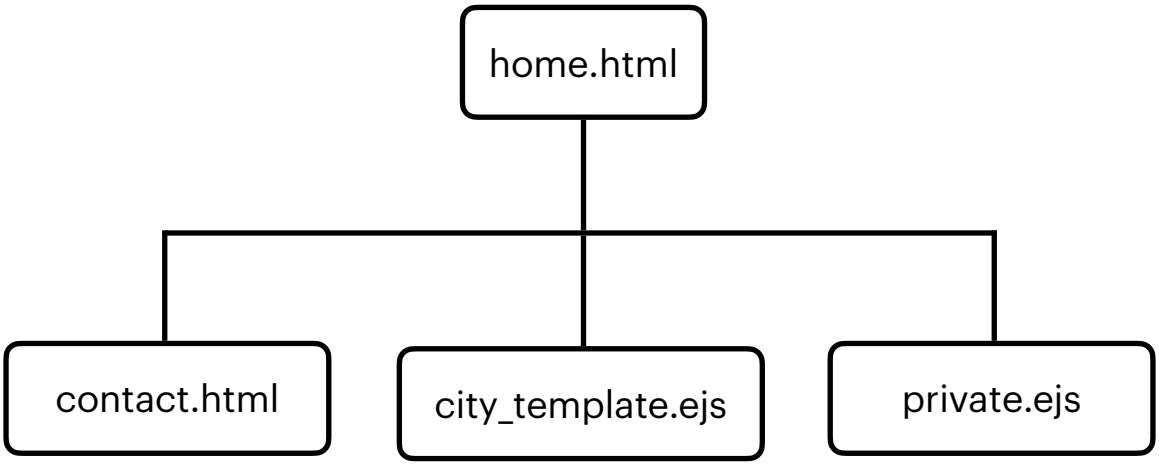
Nella pagina del meteo della singola città si trovano i dati meteorologici attuali, le previsioni per il giorno successivo e una fotografia del posto cercato, fornita da unsplash.

Nella pagina di contatto è presente un form attraverso cui l'utente, lasciando la propria email, può scrivere un feedback e inviarlo; alla pressione del tasto 'Invia' viene creata un'istanza in un database dedicato MongoDB che contiene la mail dell'utente e il feedback lasciato.

La pagina personale, costruita a seguito del login, presenta le previsioni per le successive 5 ore al momento di visualizzazione ordinate in una tabella, riferite alla città indicata durante la registrazione.



4. Diagramma dell'ordine gerarchico delle risorse



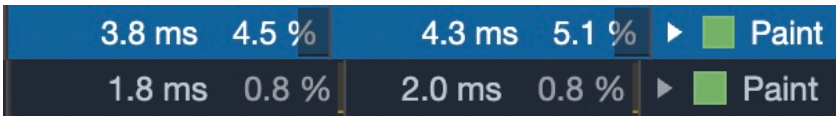
5. Ottimizzazione

Grazie all'introduzione del service worker per le chiamate API è stato ottimizzato il tempo di caricamento della home page, come si può vedere

dalle statistiche in figura, il service worker impiega fra i 700 e gli 800 ms per eseguire le richieste API, e questo tempo è risparmiato dal thread principale.

weather?lat=45.464007&units=metric&lon=9.190242&...	200	fetch	api-request_service-worker....	810 B	216 ms
weather?lat=51.502100&units=metric&lon=-0.140071...	200	fetch	api-request_service-worker....	829 B	53 ms
weather?lat=48.856386&units=metric&lon=2.295343&...	200	fetch	api-request_service-worker....	813 B	106 ms
weather?lat=40.779897&units=metric&lon=-73.96856...	200	fetch	api-request_service-worker....	806 B	52 ms
weather?lat=34.052234&units=metric&lon=-118.2436...	200	fetch	api-request_service-worker....	828 B	54 ms
weather?lat=-33.865143&units=metric&lon=151.2099...	200	fetch	api-request_service-worker....	834 B	82 ms
weather?lat=35.685013&units=metric&lon=139.75244...	200	fetch	api-request_service-worker....	813 B	148 ms

Dopo l'introduzione del webworker anche nel per la richiesta delle immagini unsplash, il tempo di painting della pagina si è complessivamente più che dimezzato, passando da 4.3ms a 2.0ms.



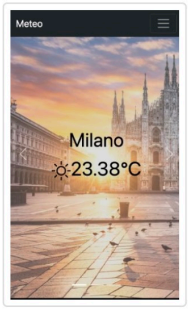
Si è inoltre adottato il formato avif per le immagini in locale, tale formato utilizza gli algoritmi di compressione AV1 e il contenitore HEIF per ottimizzare le dimensioni delle immagini, consentendo di raggiungere anche una dimensione dimezzata rispetto al formato jpeg.

Un ulteriore accortezza adottata è stata quella di attivare la compressione del testo lato server in modo da minimizzare le dimensioni dei contenuti che il server deve fornire al client; grazie a tutti questi accorgimenti le performance valutate dallo strumento lighthouse di Chrome sono passate dal 72% al 91%, se si considerano le sole performance su dispositivi mobili la percentuale sale al 99%, di seguito i dettagli delle metriche:

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 50–89 90–100



METRICS Expand view

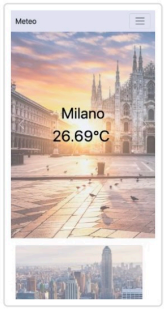
First Contentful Paint 2.2 s	Time to Interactive 2.2 s
Speed Index 3.3 s	Total Blocking Time 0 ms
▲ Largest Contentful Paint 13.0 s	Cumulative Layout Shift 0

Performance prima dell'ottimizzazione

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

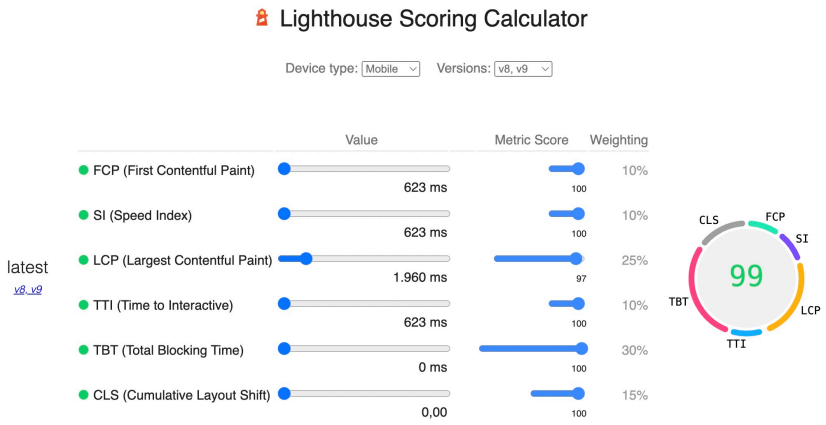
▲ 0–49 50–89 90–100



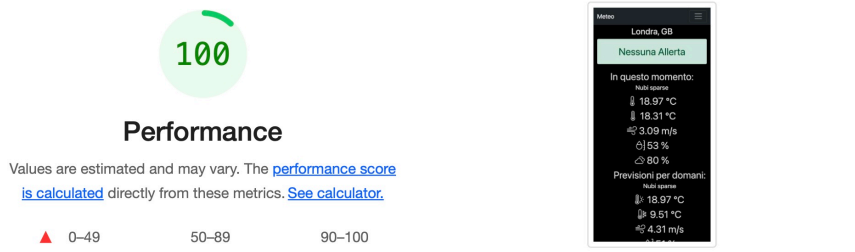
METRICS Expand view

First Contentful Paint 0.6 s	Time to Interactive 0.6 s
Speed Index 0.6 s	Total Blocking Time 0 ms
Largest Contentful Paint 2.0 s	Cumulative Layout Shift 0

Performance dopo l'ottimizzazione



Performance su dispositivo mobile



Performance di una pagina secondaria (pagina costruita dinamicamente a seguito di una ricerca)

Su tutte le pagine diverse dalla home, essendoci meno risorse da caricare, la struttura del codice garantisce delle performance che sono valutate al 100%.

6. Codice

HTML

```
<div class="carousel-inner">
    <div class="carousel-item active">
        <div class="caritem1">
          <div class="overlay bg-wh">
            <div class="city">
              Milano <br>
              <img id="mil-icon" alt="milan weather icon"><span id="mil-temp"></span>
            </div>
          </div>
        </div>
    </div>
```

Elemento del carosello nella home page, sono da notare <img> e che vengono riempiti con i dati provenienti dalle API

```
<div id="general-info">
  In questo momento:<br>
  <h5><%= description %></h5>
  <i class="bi bi-thermometer-half"></i><span> <%= temperature %> °C</span><br>
  <i class="bi bi-thermometer-high"></i><span> <%= temperature_perc %> °C</span><br>
  <i class="bi bi-wind"></i><span> <%= wind %> m/s</span><br>
  <i class="bi bi-moisture"></i><span> <%= humidity %> %</span><br>
  <i class="bi bi-clouds"></i><span> <%= clouds %> %</span>
</div>
```

Sezione nella pagina di template della città, utilizzata per la costruzione dinamica della ricerca dopo la ricerca di una qualsiasi città

```
<form action = "http://127.0.0.1:3000/tips" method = "POST">
  <div class="mb-3">
    <label for="userEmail" class="form-label">Email</label>
    <input type="email" class="form-control" id="userEmail" placeholder="name@example.com" name="us_email">
  </div>
  <div class="mb-3">
    <label for="tips" class="form-label">Comunicaci eventuali bug o nuove feature che vorresti vedere implementate
    che ti piacerebbe vedere:</label>
    <textarea class="form-control" id="tips" rows="3" name="tip"></textarea>
  </div>
  <input type="submit" class="btn btn-outline-primary" id="sub-button">
</form>
```

Form per inviare suggerimenti, quando viene premuto il tasto di submit, viene effettuata una richiesta POST al server, che provvede a memorizzare nel database locale i suggerimenti dell'utente

CSS

```css
.card > .card-img-top.ny {
  background-image: url("/img_cards/nyc.avif");
  content-visibility: auto;
  background-size: cover;
  background-position: center;
  height: 150px;
}
```

Le immagini del carosello e delle cards nella homepage sono inserite come background tramite css

```css
@media (max-width: 345px) {
  #uni {
    display: none;
  }
}
```

Media query per nascondere il tasto che reindirizza al sito unimi se la larghezza dello schermo è inferiore a 345 pixel

```css
#error {
  display: none;
  position: absolute;
  left: 50%;
  transform: translate(-50%, 0%);
  top: 0;
  margin: 15% auto;
  padding: 20px;
  border: 1px solid #888;
  width: 50%;
}
```

Banner per mostrare eventuali errori

JavaScript

```
var actualDate = new Date().getTime();
if(localStorage.getItem('darkmode')) {
  if((actualDate - JSON.parse(localStorage.getItem('darkmode')).date) > 8640000) {
      localStorage.removeItem('darkmode');
      darkModeActivation = {
        darkMode: false,
        date: new Date().getTime(),
      };
      localStorage.setItem( 'darkmode', JSON.stringify(darkModeActivation) );
    }
  } else {
    darkModeActivation = { //if there is not an option, set it to light mode
      darkMode: false,
      date: new Date().getTime(),
    };
    localStorage.setItem( 'darkmode', JSON.stringify(darkModeActivation) );
  }

if(JSON.parse(localStorage.getItem('darkmode')).darkMode){
  checkbox.checked = true;
  dark();
}
```

Quando viene caricata la pagina, viene controllato se nel local Storage dell'utente è memorizzato un oggetto 'darkmode' e se è stato aggiornato da meno di 24 ore viene applicata la dark mode, altrimenti viene resettato a light mode o, se assente, viene creato

```
async function getError() {
    var errorBanner = document.getElementById('error');
    var response = await fetch('http://127.0.0.1:3000/get_errors');
    var jsonObj = await response.json();
    if (jsonObj.errorOccured) {
      errorBanner.style.display = 'inline-block';
      errorBanner.innerText = jsonObj.errorMessage;
    }
  };
```

Al caricamento della pagina viene effettuata una richiesta al server per ottenere eventuali errori di log (password errata o sessione scaduta)

```
(async function getUserWeather () {
  navigator.geolocation.getCurrentPosition(async position => {
  const latitude = (position.coords.latitude.toFixed(6));
  const longitude = (position.coords.longitude.toFixed(6));
  const today = new Date();
  const url = `https://api.openweathermap.org/data/2.5/weather?lat=${latitude}&lon=${longitude}
  &units=metric&lang=it&appid=4da6d8179a32da39c51e22987d4e663e`;
  var response = await fetch(url);
  var jsonObj = await response.json();
  userLoc = jsonObj.name;
  weatherDes = jsonObj.weather[0].description;
  document.getElementById('userLocation').innerText = userLoc;
  document.getElementById('user-weather').innerText = weatherDes;
  document.getElementById('user-temp').innerText = jsonObj.main.temp + "°C";
  document.getElementById('user-humidity').innerText = jsonObj.main.humidity + "%";
  document.getElementById('user-wind').innerText = jsonObj.wind.speed + " m/s";
  console.log(jsonObj);

  workerPhoto.postMessage({
    userLocation : userLoc,
    weatherDescription : weatherDes,
  });
```

Viene utilizzato l'oggetto navigator per ottenere la posizione dell'utente (previo autorizzazione) e fornire i dati meteorologici per il luogo dove si trova l'utente

```
var clientID = 'Yme6ZcumIXpWryQ0DPc249CE0ua2Mxh66Y-4W2gPAAc';
var endpoint = `https://api.unsplash.com/search/photos?page=1
&query=${data.userLocation}&client_id=${clientID}`;
var photoUrl;
await fetch(endpoint)
.then(async response => {
    jsonObj = await response.json();
    console.log(jsonObj);
    if (jsonObj.total) {
        photoUrl = jsonObj.results[0].urls.thumb;
    } else {
        await fetch(`https://api.unsplash.com/search/photos?page=1
            &query=${data.weatherDescription}&client_id=${clientID}`)
        .then(async response => {
        jsonObj = await response.json();
        console.log(jsonObj);
        photoUrl = jsonObj.results[0].urls.thumb;
        })
    }
})

self.postMessage({
    'url' : photoUrl,
    'status' : 'OK',
})
```

Webworker per il caricamento dell'immagine del luogo in cui si trova l'utente, viene fatta una fetch al servizio unsplash per trovare una foto del luogo, se non si trova viene fatta una seconda fetch utilizzando la descrizione del meteo