

post

Luca Maccarini 941779

Relazione progetto

Programmazione web e mobile

Università degli studi di Milano

2021/2022

Cloudy

Indice

Destinatari e Analisi dei requisiti	2
1.1 Requisiti funzionali	2
1.2 Requisiti non funzionali	3
Requisiti di sviluppo	3
Requisiti di sicurezza	3
interfacce	4
Architettura	11
3.1. Architettura delle risorse	11
3.2. Flusso dei dati	12
3.2. Architettura del codice	14
3.3. Frammenti significativi di codice	15
Valutazione delle prestazioni	21
Conclusioni	23

1. Destinatari e Analisi dei requisiti

É richiesta la realizzazione di un'applicazione web per offrire un servizio meteorologico agli utenti, curiosi di sapere il meteo di una certa località.

Il target utenti è di qualsiasi età e stato sociale, per questo per permettere a tutto il target la possibilità di accedere al servizio, l'applicazione web dovrà essere utilizzabile su dispositivi eterogenei dotati di un browser ed una connessione ad internet.

La piattaforma dovrà mostrare le previsioni meteo tenendo conto della giornata corrente (ovviamente le ore passate possono essere trascurate) e delle successive 3 o 4 giornate. Le previsioni di una giornata dovranno essere riportate ad intervalli di 3 ore così che l'utente possa avere maggiore chiarezza di come sarà il meteo durante tutta la giornata.

Un aspetto importante è l'esperienza utente, ovvero per l'utente dovrà essere piacevole utilizzare l'applicazione in termini di grafica, affidabilità e performance.

Tipologia di utenti

Utilizzando il modello di Marcia J. Bates mi aspetto che la piattaforma sia utilizzata da utenti diretti attivi o passivi, ovvero da persone che vogliono cercare il meteo di una località e lo fanno in maniera attiva o passiva grazie ad una notifica inviata dalla applicazione web.

	Active	Passive
Directed	Searching	Monitoring
Undirected	Browsing	Being Aware

Per questo sarà fondamentale implementare:

- una barra di ricerca per cercare il meteo della località (utenti diretti-attivi)
- un form per l'iscrizione ad una newsletter per ricevere ogni mattina il meteo della giornata riferito ad una o più località (utenti diretti-passivi)

Requisiti di sistema

1.1 Requisiti funzionali

- Cercare il meteo di una località
- Visualizzare il meteo della località in cui ci si trova (geolocalizzando l'utente)

- iscriversi ad una newsletter per ricevere le info sul meteo della giornata ogni mattina
- iscriversi a più newsletter, nel caso in cui ogni mattina si voglia il meteo della giornata di più località
- selezionare il giorno preciso del quale si vuole vedere il meteo dettagliato (ogni 3 ore)

1.2 Requisiti non funzionali

Requisiti di sviluppo

Sono previsti dei requisiti minimi per l'applicazione web (scelti dal professore):

- Le pagine devono essere sviluppate in formato HTML5
- Tutte le pagine devono essere validate
- Il layout delle pagine deve essere sviluppato con CSS
- L'applicazione dovrà servirsi di almeno una API HTML5
- Il progetto deve implementare una o più chiamate XMLHttpRequest e le chiamate possono interrogare dati in JSON, XML, XHTML, TXT.
- Il progetto deve implementare una o più chiamate a un servizio NodeJS sviluppato dallo studente e le chiamate devono interrogare o caricare dati in JSON o XML.

inoltre per la memorizzazione di dati aggiungo:

- utilizzo del database NoSQL mongoDB

Requisiti di sicurezza

- verifica degli input inseriti dall'utente mediante l'interfaccia, per assicurarsi di non elaborare input malevoli o non significativi;
- eseguire backup periodici del database per permettere il ripristino in caso di problemi gravi e limitare le perdite di dati.

Qualità del software da raggiungere

- **Facilità di modifica**
Rendere il codice strutturato per identificare con facilità gli errori, realizzare una struttura che permetta di aggiungere nuove funzionalità per sviluppi futuri.
- **Portabilità**
Il sistema deve funzionare su diverse piattaforme, così da permettere agli utenti con dispositivi eterogenei di utilizzare il servizio.
- **Riusabilità**
Rendere il codice modulare e dove possibile generico per poter riutilizzare parti del codice per la creazione di nuove applicazioni web.

- **Usabilità**

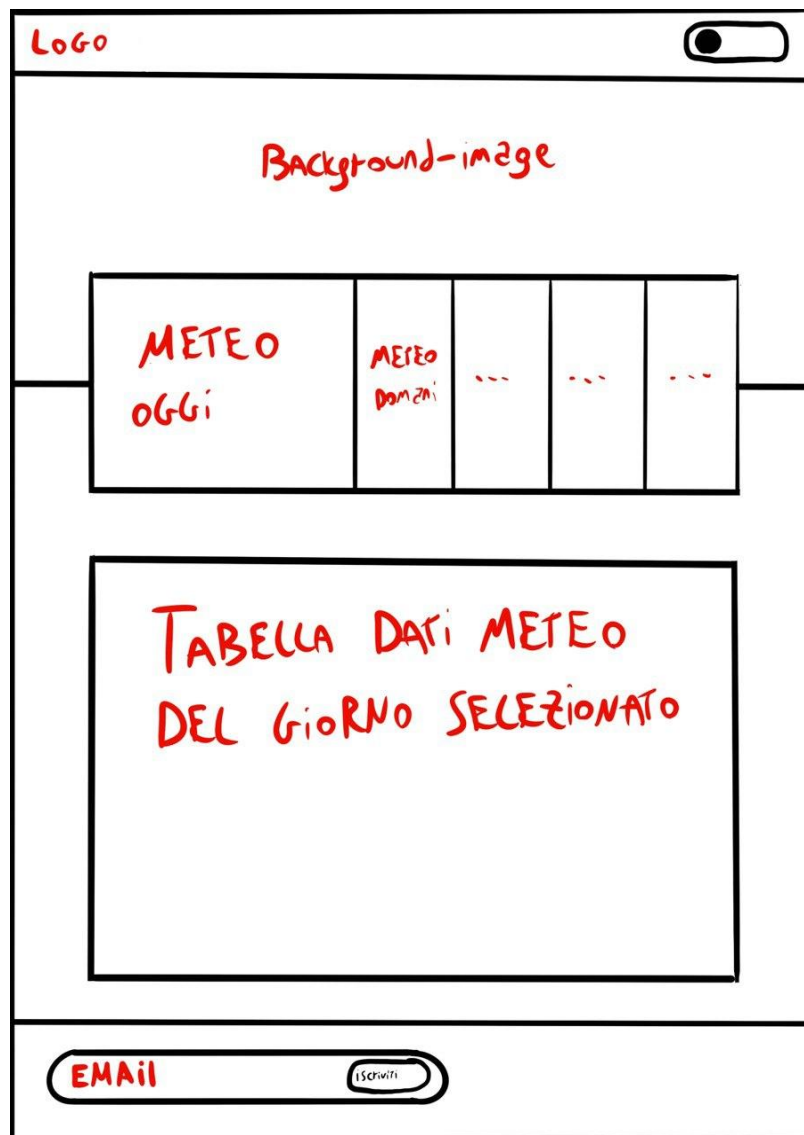
E' importante che per gli utenti sia confortevole e performante utilizzare l'applicazione in termini grafici e di performance

2. interfacce

L'intera applicazione web si sviluppa su una singola pagina "index.html" che viene opportunamente manipolata dal javascript allegato. Esiste anche una seconda pagina "subscribed_newsletter.ejs" che viene utilizzata solo per mostrare l'esito della registrazione alla newsletter

Mockup iniziale disegnato a mano

per farmi una idea iniziale di che forma dare all'interfaccia ho steso questo breve disegno



successivamente ho prodotto la struttura della pagina in html e ho utilizzato bootstrap più del css scritto da me per realizzare lo stile e la responsività della pagina. Seguono degli screenshots dell'interfaccia

Pagina iniziale con geolocalizzazione rifiutata



Pagina iniziale con geolocalizzazione accettata

The screenshot displays the 'Cloudy' weather application interface. At the top, the 'Cloudy' logo and a toggle switch are visible. Below is a search bar with the placeholder text 'Cerca la tua città' and a 'Cerca' button. The main content area features a large weather card for 'Verdello' showing a current temperature of 31°C and a sun with clouds icon. To the right of this card is a 5-day forecast grid. Below the main card is a section titled 'Dati del giorno' containing a table of hourly weather data. At the bottom of the page is a dark footer area with a newsletter sign-up form and a copyright notice.

Verdello
31°C

Martedì	5 Lug	Mercoledì	Giovedì	Venerdì	Sabato
Verdello					
31°C		27°C	29°C	27°C	27°C

Dati del giorno

Orario	Temp. °C	Percepita °C	Min. °C	Max. °C	Icona	Descrizione	Umidità %	Vento m/s
20:00	33	33	31	33		poche nuvole	37	1.81
23:00	28	29	26	28		nubi sparse	47	3.27



per ricevere ogni giorno il meteo di Verdello iscriviti alla nostra news letter

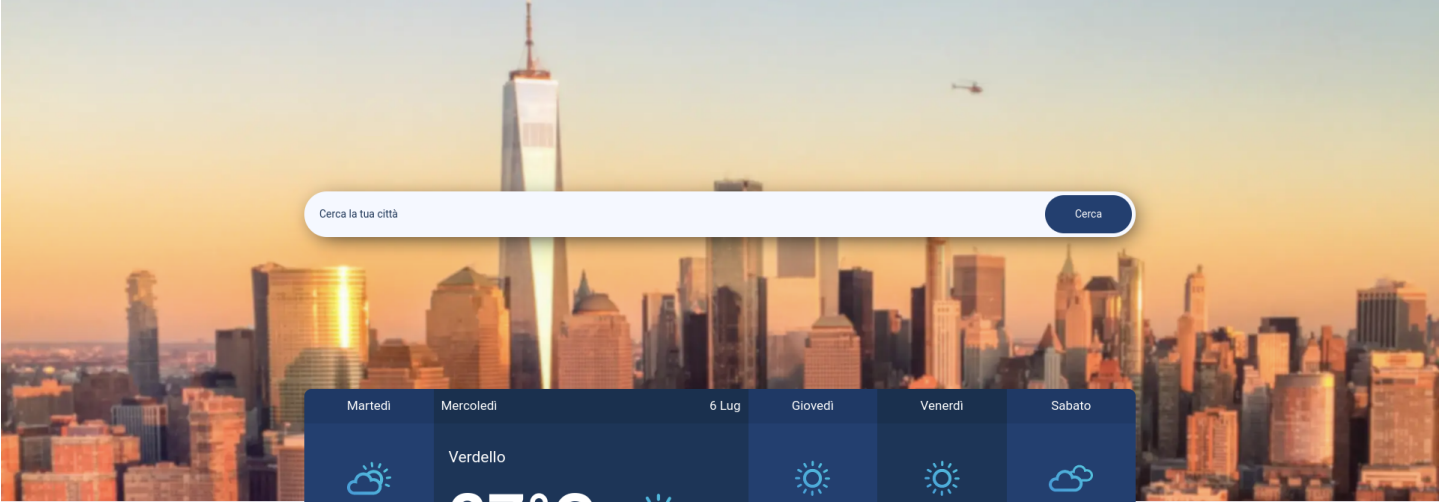
Inserisci la tua email

Copyright 2022 Cloudy. Designed by Luca Maccarini. All rights reserved


Dopo aver accettato la geolocalizzazione viene eseguita una richiesta ajax (da un web worker) che chiede al server NodeJS i dati meteo della località in questione, dopo di che vengono aggiornati i dati presenti nella schermata e vengono mostrati. (i dati del giorno corrente comprendono solo le successive ore al caricamento della pagina)

pagina iniziale con geolocalizzazione accettata e dopo aver cliccato sul prossimo giorno






Martedì



31°C

Mercoledì


Verdello

27°C 


6 Lug


29°C


Giovedì


27°C

Venerdì


27°C

Sabato


27°C

Dati del giorno

Orario	Temp. °C	Percepita °C	Min. °C	Max. °C	Icona	Descrizione	Umidità %	Vento m/s
2:00	20	20	20	20		pioggia moderata	87	2.92
5:00	18	19	18	18		pioggia moderata	89	1.98
8:00	21	21	21	21		pioggia leggera	81	0.78
11:00	26	26	26	26		cielo sereno	55	1.04
14:00	30	30	30	30		cielo sereno	42	0.38
17:00	31	31	31	31		cielo sereno	38	1.29
20:00	28	29	28	28		pioggia leggera	50	2.26
23:00	23	23	23	23		cielo sereno	65	2.53

per ricevere ogni giorno il meteo di Verdello iscriviti alla nostra news letter

Copyright 2022 Cloudy. Designed by Luca Maccarini. All rights reserved

Cliccando i vari giorni presenti nel widget è possibile selezionare il giorno che si vuole vedere, così di conseguenza la tabella sottostante viene aggiornata dal javascript.

Cercando una località nella barra di ricerca e cliccando il pulsante “cerca” verrà eseguita una richiesta ajax (da un web worker) che richiede al server NodeJS i dati meteo della nuova località, dopo di che vengono aggiornati i dati presenti nella schermata.

Pagina iniziale visualizzata da un dispositivo mobile



L'applicazione web è responsive grazie alle librerie di bootstrap, come si vede il widget in alto diventa più compatto e nella tabella con i vari dati meteo è possibile scorrere da destra a sinistra per mostrare i dati che nell'immagine sono nascosti

Pagina iniziale con il tema scuro

The screenshot shows the home page of the 'Cloudy' weather app in dark theme. At the top, there's a navigation bar with the 'Cloudy' logo and a light/dark toggle switch. Below the navigation bar is a search bar with the placeholder text 'Cerca la tua città' and a 'Cerca' button. The main content area displays the weather for 'Verdello' on 'Martedì 5 Lug'. The current temperature is 31°C, and the forecast for the next four days (Mercoledì, Giovedì, Venerdì, Sabato) shows temperatures of 27°C, 29°C, 27°C, and 27°C respectively. Below this is a section titled 'Dati del giorno' which contains a table with hourly weather data for 20:00 and 23:00. At the bottom, there's a newsletter sign-up section with the text 'per ricevere ogni giorno il meteo di Verdello iscriviti alla nostra news letter', an input field for the email, and an 'iscriviti' button. The footer contains the copyright notice: 'Copyright 2022 Cloudy. Designed by Luca Maccarini. All rights reserved'.

Cloudy

Cerca la tua città

Cerca

Martedì 5 Lug

Verdello

31°C

Temperatura media (tra le ore: 20:00 - 23:00)

Mercoledì 27°C

Giovedì 29°C

Venerdì 27°C

Sabato 27°C

Dati del giorno

Orario	Temp. °C	Percepita °C	Min. °C	Max. °C	Icona	Descrizione	Umidità %	Vento m/s
20:00	33	33	31	33		poche nuvole	37	1.81
23:00	28	29	26	28		nubi sparse	47	3.27

per ricevere ogni giorno il meteo di Verdello iscriviti alla nostra news letter

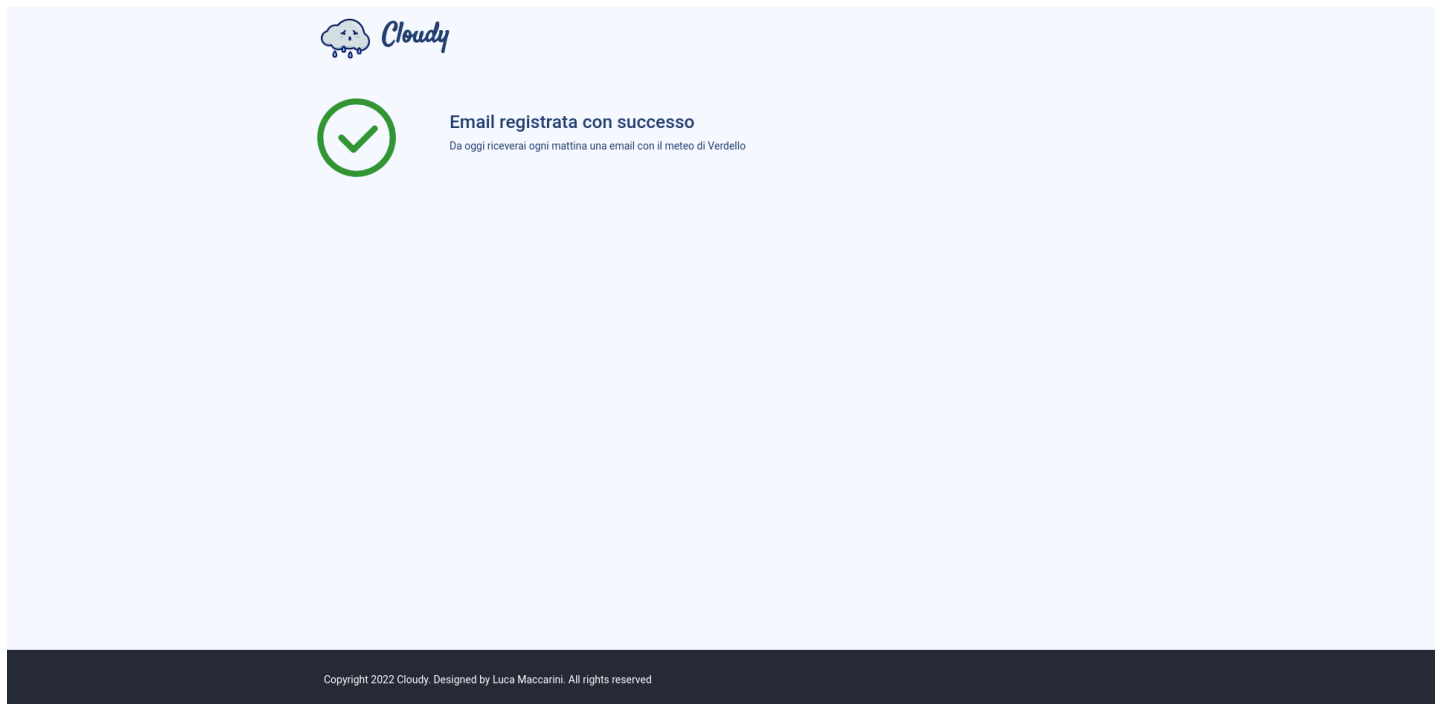
Inserisci la tua email

iscriviti

Copyright 2022 Cloudy. Designed by Luca Maccarini. All rights reserved

Cliccando la lampadina in alto a destra è possibile cambiare il tema della pagina. La preferenza del tema rimane salvata nel localStorage così da mantenere lo stesso tema quanto l'utente tornerà sulla piattaforma.

Pagina di avvenuta iscrizione alla newsletter



Newsletter



screenshots? meglio provare in prima persona!

invito a vedere e a provare personalmente la piattaforma per avere una migliore idea di come è l'interazione con l'utente e valutarne l'esperienza utente.

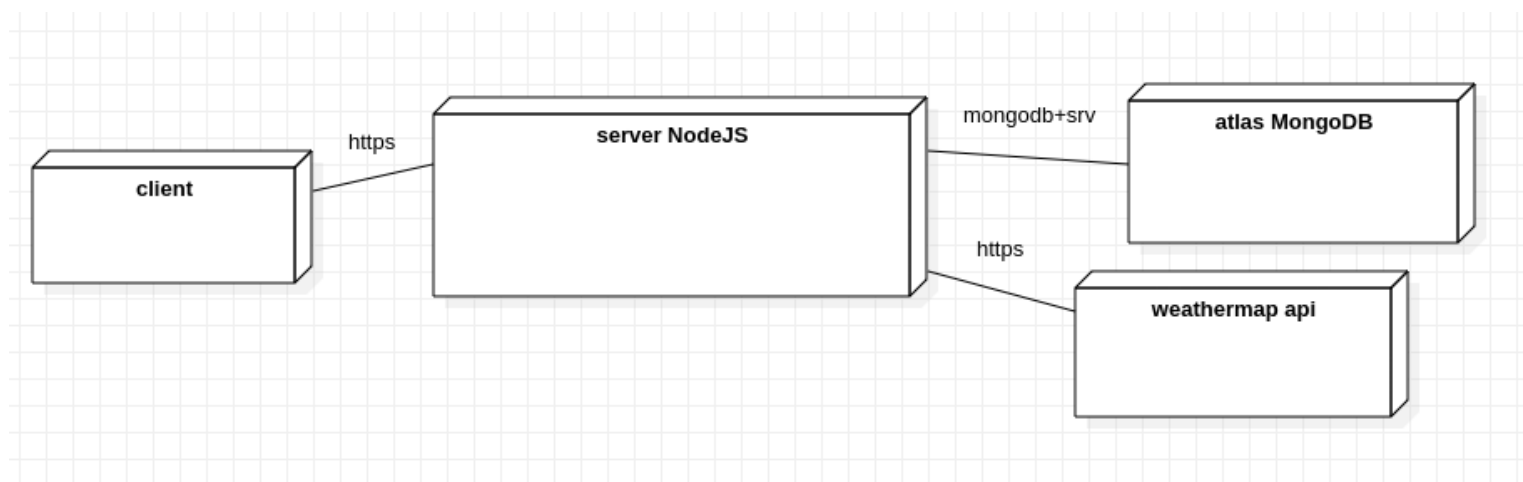
La piattaforma è hostata su un server Heroku ed è disponibile al link:

<https://cloudymaccarini.herokuapp.com/>

(N.B. il server heroku è un server gratuito con basse prestazioni)

3. Architettura

Riporto un diagramma di deployment per mostrare come sono interconnessi i vari nodi interessati e che protocollo viene utilizzato nelle varie comunicazioni



3.1. Architettura delle risorse

l'architettura delle risorse rispetta quanto visto a lezione con il framework express, ovvero:

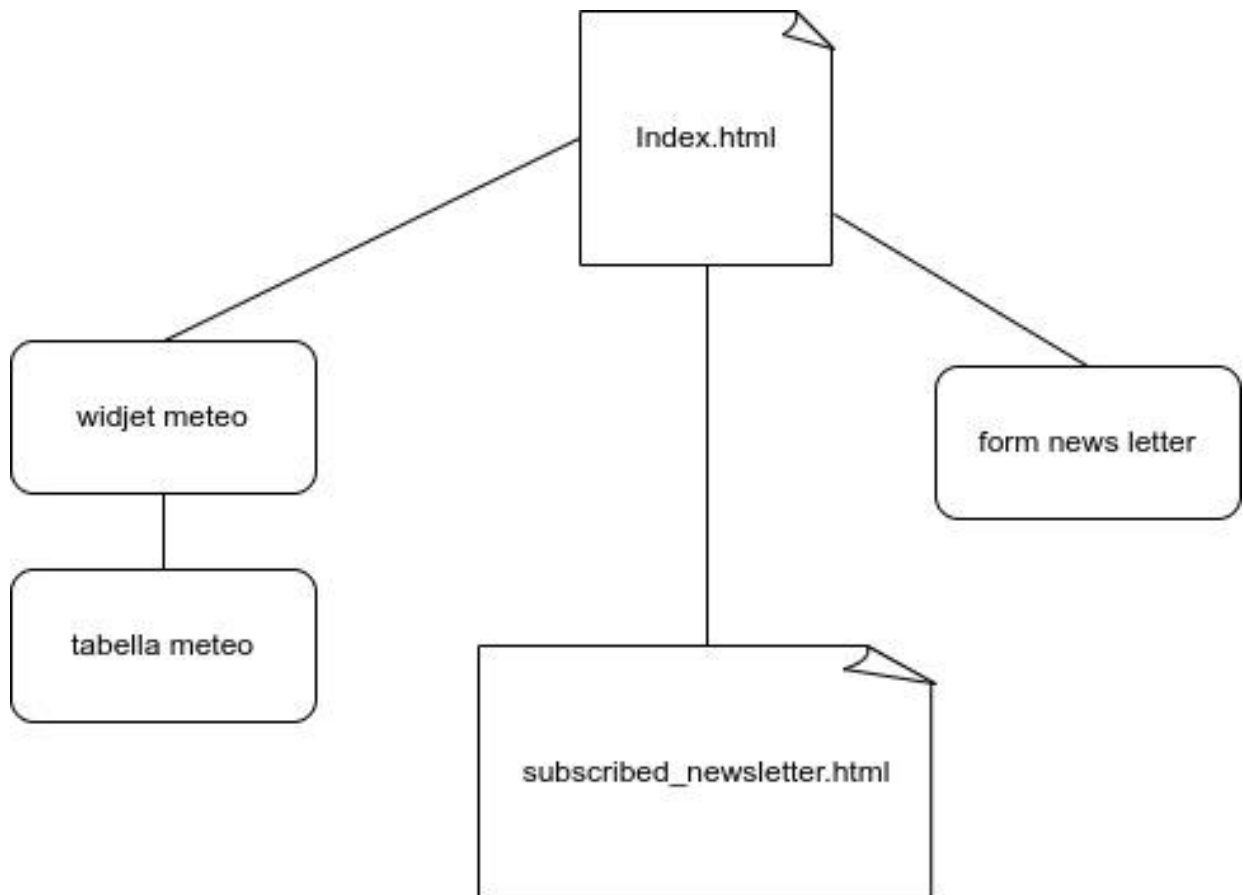
- la cartella "assets": contiene tutti i file statici divisi nelle cartelle "css", "font", "immagini" e "javascript"
- la cartella "views": contiene tutti i template html quindi le varie pagine che devono essere servite al client, inoltre questa cartella contiene anche il template della newsletter
- la cartella "node_modules": contiene tutti i moduli utilizzati dal server NodeJS
- la cartella "newsletter_email_sender": contiene in file dove è presente il codice per inviare le email

inoltre viene anche rispettato il design pattern Model-View-Controller in quanto:

- la pagina principale index.html è la view
- il javascript allegato nella pagina principale è il controller, infatti si occupa di gestire gli input dell'utente, a fare le chiamate ajax e a modificare la view

- il model sono gli oggetti contenenti i dati meteo ricevuti (in formato json) dal servizio api

l'applicazione si basa su una sola pagina principale contenente un widget che indica in maniera generica il meteo ed una tabella sottostante con informazioni più specifiche. Infine nel footer c'è una form per l'iscrizione alla newsletter del meteo



3.2. Flusso dei dati

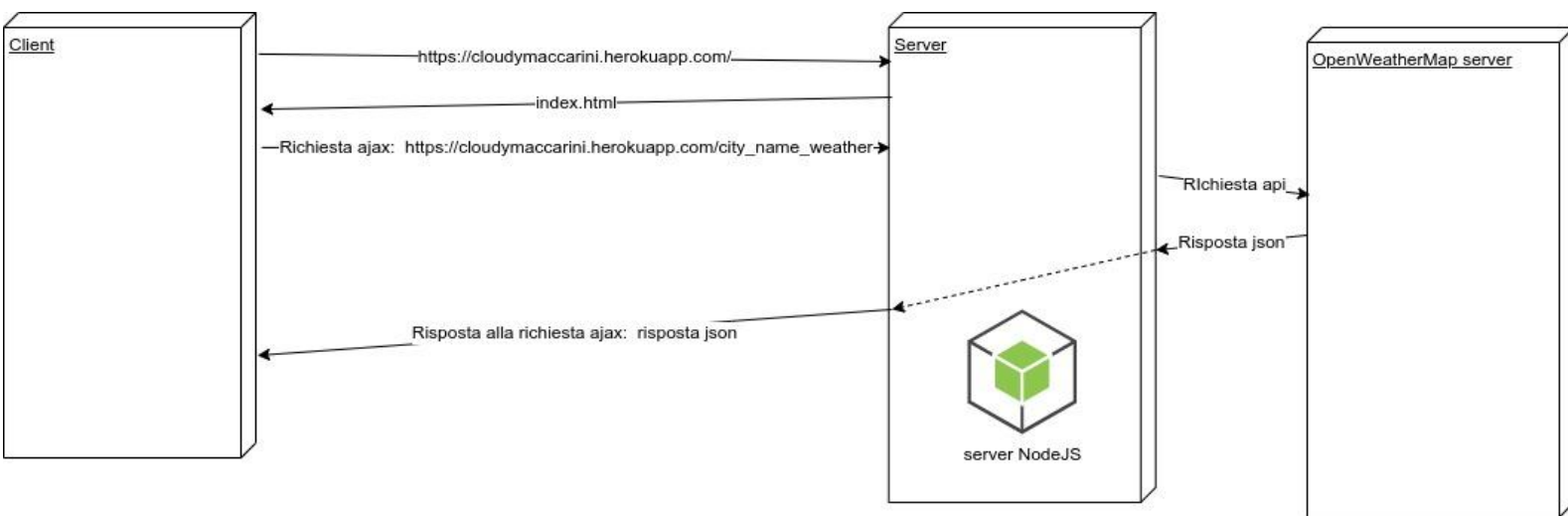
Dati meteo

Quando l'utente naviga sulla pagina principale e viene geolocalizzato dal javascript oppure ricerca una località mediante la barra di ricerca: un web worker esegue una richiesta ajax al server NodeJS il quale trasforma questa richiesta in una richiesta api al servizio di OpenWeatherMap. Una volta che il servizio api risponde al server NodeJs (la risposta è un file json), la risposta viene inoltrata al javascript che aveva fatto la richiesta ajax. Dopo aver ottenuto i dati del meteo questi vengono trasformati, infatti, il json viene dato ad un web worker incaricato di fare un parsing del json e a calcolare vari dati come:

- la temperatura media di ogni giornata
- la frequenza assoluta dell'icona meteo di ogni giornata.

Infine i dati meteo ed i dati calcolati vengono mostrati nella pagina.

Riporto un diagramma che mostra l'interazione dei nodi quando un utente ricerca una località



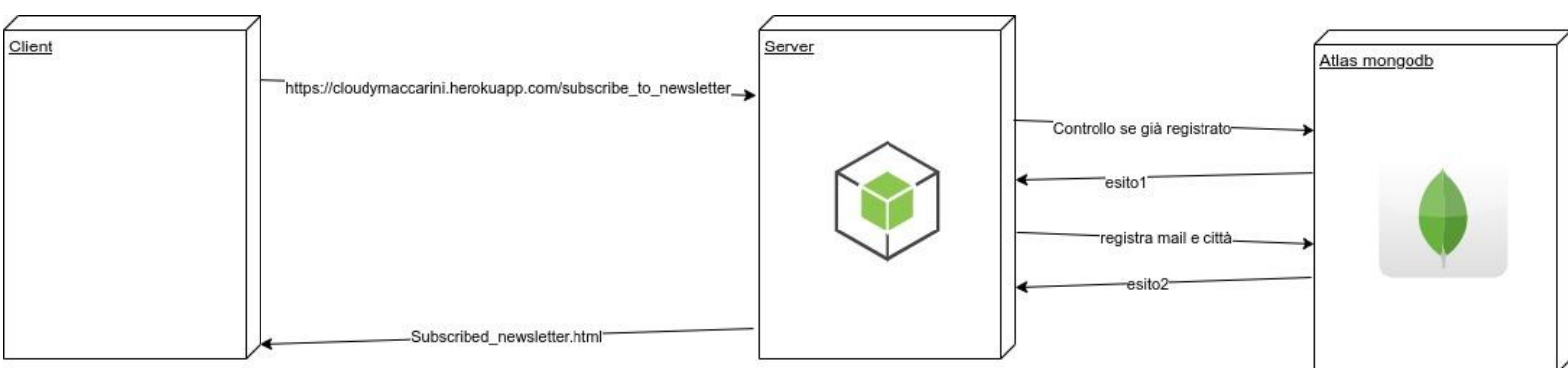
quando un utente invece viene geolocalizzato lo schema è praticamente identico cambia solo che la richiesta ajax sarà fatta all'indirizzo https://cloudymaccarini.herokuapp.com/lon_lat_weather e viene passato in post la latitudine e la langitudine

Una domanda che potrebbe sorgere spontanea: “come mai passiamo per il server NodeJS per ottenere i dati meteo? la richiesta ajax potrebbe interfacciarsi direttamente con il servizio api giusto?”

Il problema è che non si deve esporre la api key fornita dal servizio meteo, infatti la api key è salvata sul server ed essendo necessaria per fare le chiamate api questo rende il server l'unico in grado di fare tali richieste. Se il javascript sul client fosse a conoscenza della api key un utente malevolo sarebbe in grado di estrarla ed utilizzarla a suo piacimento.

Iscrizione alla newsletter

Quando l'utente compila la form situata nel footer della `index.html` e clicca il bottone di iscrizione i dati (email e località) della form vengono inviati al server NodeJS mediante una richiesta POST. Il server provvederà prima di tutto a controllare se l'utente ha già una newsletter registrata per la medesima località e se non la ha il server procede alla registrazione della email e della località nel database mongoDB



Nel diagramma ho dato per scontato che quando il server controlla se l'utente ha già una email registrata per la località in questione abbia esito negativo così da poter procedere alla registrazione. Ho fatto questa assunzione solo perché mi interessava mostrare solo la comunicazione tra i vari nodi.

3.2. Architettura del codice

ho diviso il codice javascript in file basandomi sul compito ricoperto dal codice:

- **display_weather_data.js**: si occupa di aggiornare i dati presenti nella index.html
- **email_validator.js**: controlla il formato dell'email inserita, questo file è utilizzato sia lato server che lato client
- **geolocation.js**: geolocalizza l'utente e ottenere i dati meteo mediante il web worker "**weather_request_worker.js**" che contiene il codice per la richiesta ajax
- **get_weather.js**: attende che "**weather_request_worker.js**" ottiene i dati meteo dal server NodeJs e successivamente mediante il worker "**api_data_parser.js**" effettua il parsing del json
- **image_lazy_loader_script.js**: permette di caricare in background le immagini da renderizzare grazie al worker "**image_loader.js**"
- **search_button.js**: si occupa di ottenere i dati meteo della località cercata mediante il worker "**weather_request_worker.js**" che contiene il codice per la richiesta ajax
- **white_dark_theme_changer.js**: si occupa di cambiare il tema e salvarlo nel local storage

- **white_dark_theme_loader.js**: si occupa di caricare il tema
- **server.js**: contiene tutto il codice del server e il codice che serve ad inviare le newsletter ogni mattina utilizzando il file “**send_mail.js**”

3.3. Frammenti significativi di codice

“server.js” richieste al servizio api

```

52 app.post('/city_name_weather', urlencodedParser, function(req, res) {
53   let city = req.body.city;
54   axios.get('https://api.openweathermap.org/geo/1.0/direct?q=' + city + '&limit=1&appid=' + apikey)
55   .then(axios_res => {
56     let lat = axios_res.data[0].lat;
57     let lon = axios_res.data[0].lon;
58     axios.get('https://api.openweathermap.org/data/2.5/forecast?lat=' + lat + '&lon=' + lon + '&appid=' + apikey + '&units=metric&lang=it')
59     .then(axios_res => {
60       res.json(axios_res.data);
61     })
62     .catch(error => {
63       //la richiesta api (con le coordinate) ha dato esito negativo
64       res.status(500).send({errore: "meteo lon, lat", message: error.data});
65     });
66   })
67   .catch(error => {
68     //la richiesta api (ricerca città) ha dato esito negativo
69     res.status(500).send({errore: "ricerca città", message: error.data});
70   });
71 });
72
73
74
75 });
76
77 app.post('/lon_lat_weather', urlencodedParser, function(req, res) {
78   let lat = req.body.lat;
79   let lon = req.body.lon;
80   axios.get('https://api.openweathermap.org/data/2.5/forecast?lat=' + lat + '&lon=' + lon + '&appid=' + apikey + '&units=metric&lang=it')
81   .then(axios_res => {
82     res.json(axios_res.data);
83   })
84   .catch(error => {
85     res.status(500).send(JSON.stringify({errore_api_weather: error.data}));
86   });
87 });
88
89

```


“server.js” registrazione alla newsletter (interazione con mongodb)

```
94 app.post('/subscribe_to_newsletter', urlencodedParser, function(req, res) {
95   let email = req.body.email;
96   let city = req.body.city;
97
98   if(!validateEmail(email)){
99     res.render("subscribed_newsletter", {img_src: "not_ok", titolo:"Non è stato possibile registrarti", messaggio:"La email inserita non è una vera email"})
100     return;
101   }
102
103
104   MongoClient.connect(
105     mongo_db_url,
106
107     {
108       useNewUrlParser: true,
109       useUnifiedTopology: true
110     },
111
112     (err, client) => {
113       if(err) {
114         res.render("subscribed_newsletter", {img_src: "not_ok", titolo:"Non è stato possibile registrarti", messaggio:"Errore durante la connessione al database"})
115       }else{
116         const db = client.db(process.env.db_name);
117         const city_email_collection = db.collection("city_email");
118
119         city_email_collection.findOne({email: email, city:city}, (err, result) =>{
120           if(err){
121             res.render("subscribed_newsletter", {img_src: "not_ok", titolo:"Non è stato possibile registrarti", messaggio:"Errore della collection"})
122           }else{
123             if(!result){
124               city_email_collection.insertOne({email: email, city: city}, (err, result) => {})
125               res.render("subscribed_newsletter", {img_src: "ok", titolo:"Email registrata con successo", messaggio:"Da oggi riceverai ogni mattina una email con il meteo di " + city})
126             }else{
127               res.render("subscribed_newsletter", {img_src: "ok", titolo:"La tua email è già registrata", messaggio:"Dovresti già ricevere le email inerenti al meteo di " + city})
128             }
129           }
130         })
131       }
132     })
133   }
134 })
135 });
136
137
```

“get_weather.js e weather_request_worker.js” richiesta ajax con web worker

```
1  const weather_request_worker = new Worker('js/workers/weather_request_worker.js');
2
3  weather_request_worker.onmessage = function(e){
4    if("ajax_error" in e.data){
5      if(e.data.ajax_error.errorre == "ricerca città")
6        alert("la città inserita non è stata trovata");
7      else{
8        alert("latitudie o longitudine non valide");
9      }
10   } else {
11     parse_api_data(e.data);
12   }
13 }
```

```

14 self.onmessage= function(e){
15     let xhr = new XMLHttpRequest();
16     xhr.responseType = 'json';
17
18     if('city' in e.data){
19         var params = 'city=' + e.data.city;
20         xhr.open('POST', '/city_name_weather', true);
21     }else{
22         var params = 'lat=' + e.data.lat + "&lon=" + e.data.lon;
23         xhr.open('POST', '/lon_lat_weather', true);
24     }
25
26     xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
27
28     xhr.onload = function() {
29         if(xhr.status == 200) {
30             self.postMessage(this.response);
31         }else{
32             self.postMessage({ajax_error:this.response});
33         }
34     }
35     xhr.send(params);
36 }

```

Ho utilizzato un web worker per fare la richiesta ajax perchè sarebbe stato poco performante dare tale compito al thread principale il quale avrebbe dovuto aspettare molto tempo inutilmente.

“white_dark_theme_changer.js” utilizzo del local storage per salvare il tema

```
3 var dark_white_checkbox = document.createElement('input');
4 dark_white_checkbox.type = 'checkbox';
5 dark_white_checkbox.id = "dark-light-mode-checkbox"
6 dark_white_checkbox.className=""
7
8 if(start_white_theme === 'true' || start_white_theme===null)
9     dark_white_checkbox.checked = true
10 else
11     dark_white_checkbox.checked = false
12
13 dark_white_checkbox.addEventListener("change", function(){
14     if(dark_white_checkbox.checked){
15         document.body.classList.remove("bootstrap-dark");
16         document.body.classList.add("bootstrap");
17         localStorage.setItem('white_theme', true);
18     }
19     else{
20         document.body.classList.remove("bootstrap");
21         document.body.classList.add("bootstrap-dark");
22         localStorage.setItem('white_theme', false);
23     }
24 });
25
26 document.getElementById("dark-light-mode-div").appendChild(dark_white_checkbox);
27
```

“image_lazy_load_script.js e image_loader” web worker per il caricamento delle immagini in background

```
2  const ImageLoaderWorker = new Worker('js/workers/image_loader.js')
3
4  ImageLoaderWorker.addEventListener('message', function(e){
5
6      let imageData = e.data
7
8      let objectURL = URL.createObjectURL(imageData.blob)
9
10     if(imageData.is_background == true){
11         let imageElements = document.querySelectorAll(`*[data-background-image='${imageData.imageURL}']`)
12         imageElements.forEach(imageElement => {
13             imageElement.style.backgroundImage = "url('"+objectURL+"')"
14             imageElement.removeAttribute('data-src')
15         })
16     }else{
17         let imageElements = document.querySelectorAll(`img[data-src='${imageData.imageURL}']`)
18         imageElements.forEach(imageElement => {
19             imageElement.setAttribute('src', objectURL)
20             imageElement.removeAttribute('data-src')
21
22             if(imageElement.classList.contains("show_after_load"))
23                 imageElement.classList.remove("show_after_load");
24         })
25     }
26 }
27 })
28
29
30 function load_images(){
31     let imageURLSet = new Set();
32     let imgElements = document.querySelectorAll('img[data-src]')
33     imgElements.forEach(imageElement => {
34         let imageURL = imageElement.getAttribute('data-src')
35
36         if(!imageURLSet.has(imageURL)){
37
38             ImageLoaderWorker.postMessage({
39                 is_background: false,
40                 url: imageURL,
41             })
42
43             imageURLSet.add(imageURL);
44         }
45     })
46 }
47 }
```

il caricamento in background delle immagini è stato visto a lezione, ma il codice che ho utilizzato non è identico a quello in classe infatti come si vede alla riga 31 ho introdotto un Set in modo da memorizzare tutti gli url caricati così in caso di immagini ripetute non avrei ricaricato l'intera immagine ma avrei utilizzato quella già caricata precedentemente. Questo è stato molto utile soprattutto con le icone del

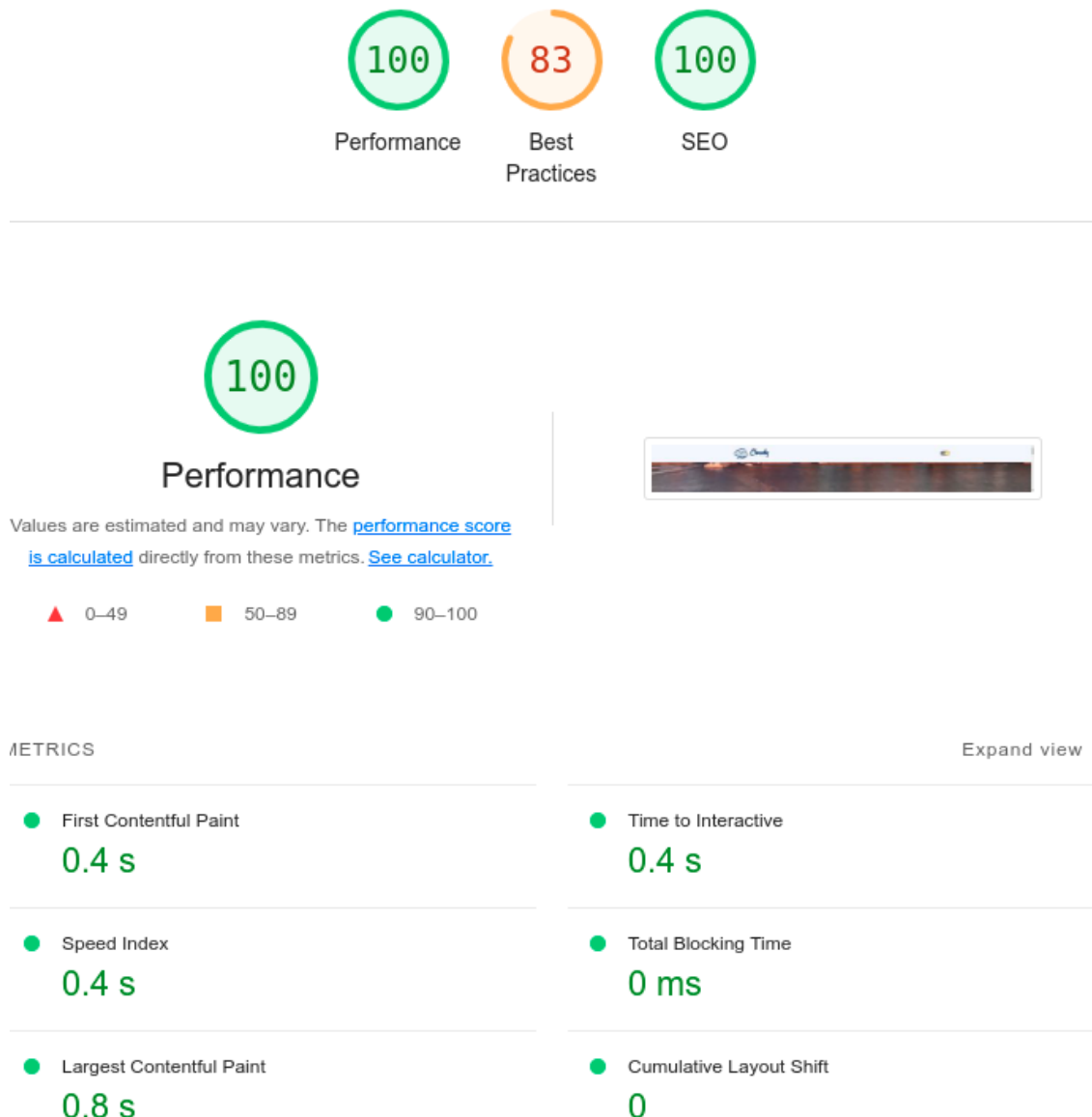
meteo che vengono ripetute parecchie volte tenendo conto che al caricamento della pagina vengono caricate tutte le icone meteo dei 3 o 4 giorni successivi più alle icone del widget.

```
2  self.addEventListener('message', async function(e){
3      let imageURL = e.data.url
4
5      let response = await fetch(imageURL)
6      let blob = await response.blob()
7
8      self.postMessage({
9          is_background: e.data.is_background,
10         imageURL: imageURL,
11         blob: blob,
12     })
13 })
```

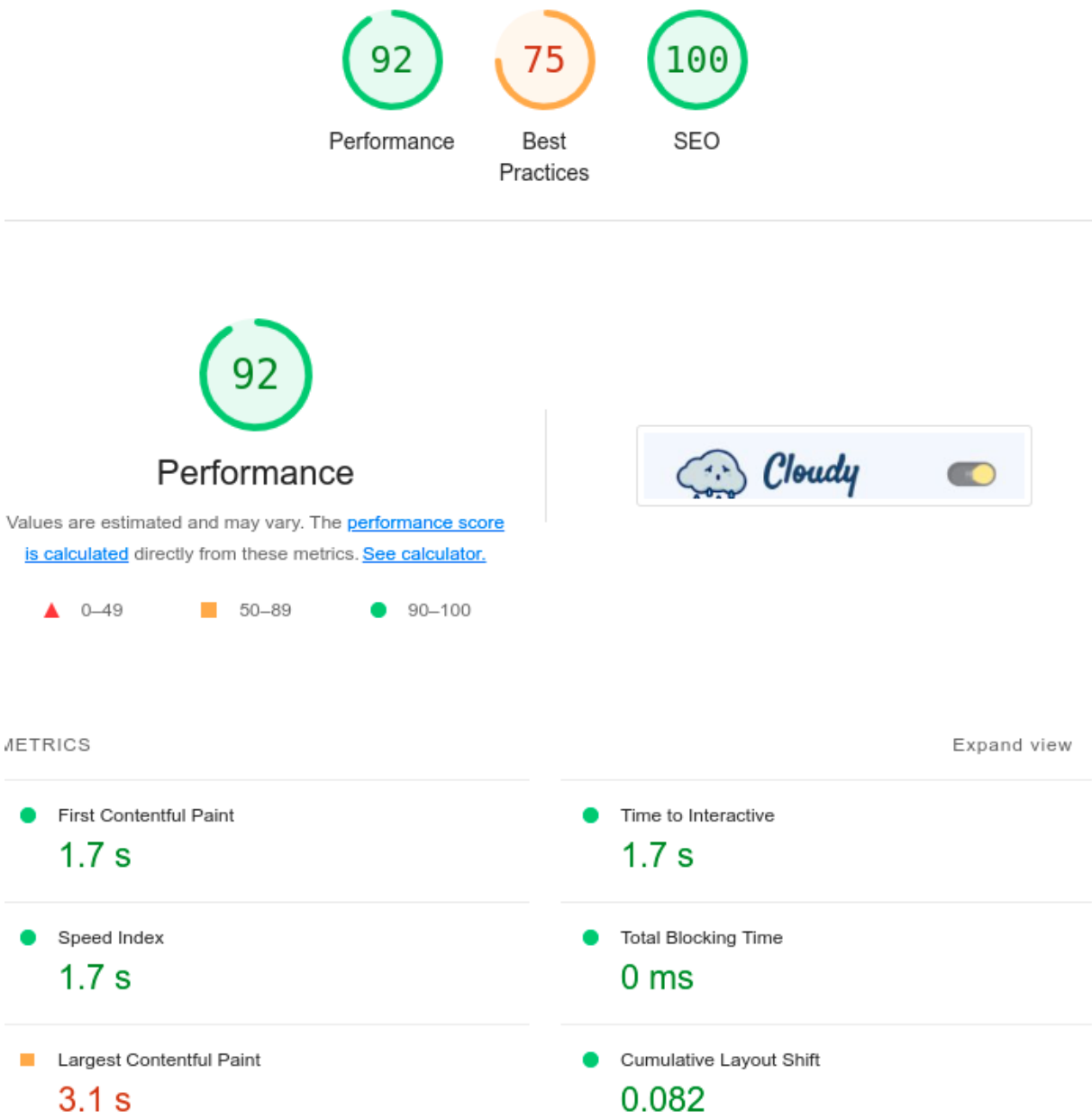
4. Valutazione delle prestazioni

Nota Iniziale: la valutazione delle prestazioni è stata fatta in locale con lighthouse in quanto il server sul quale è hostata l'applicazione è un server gratuito e con prestazioni scarse, quindi avrei avuto dei risultati falsati.

Desktop

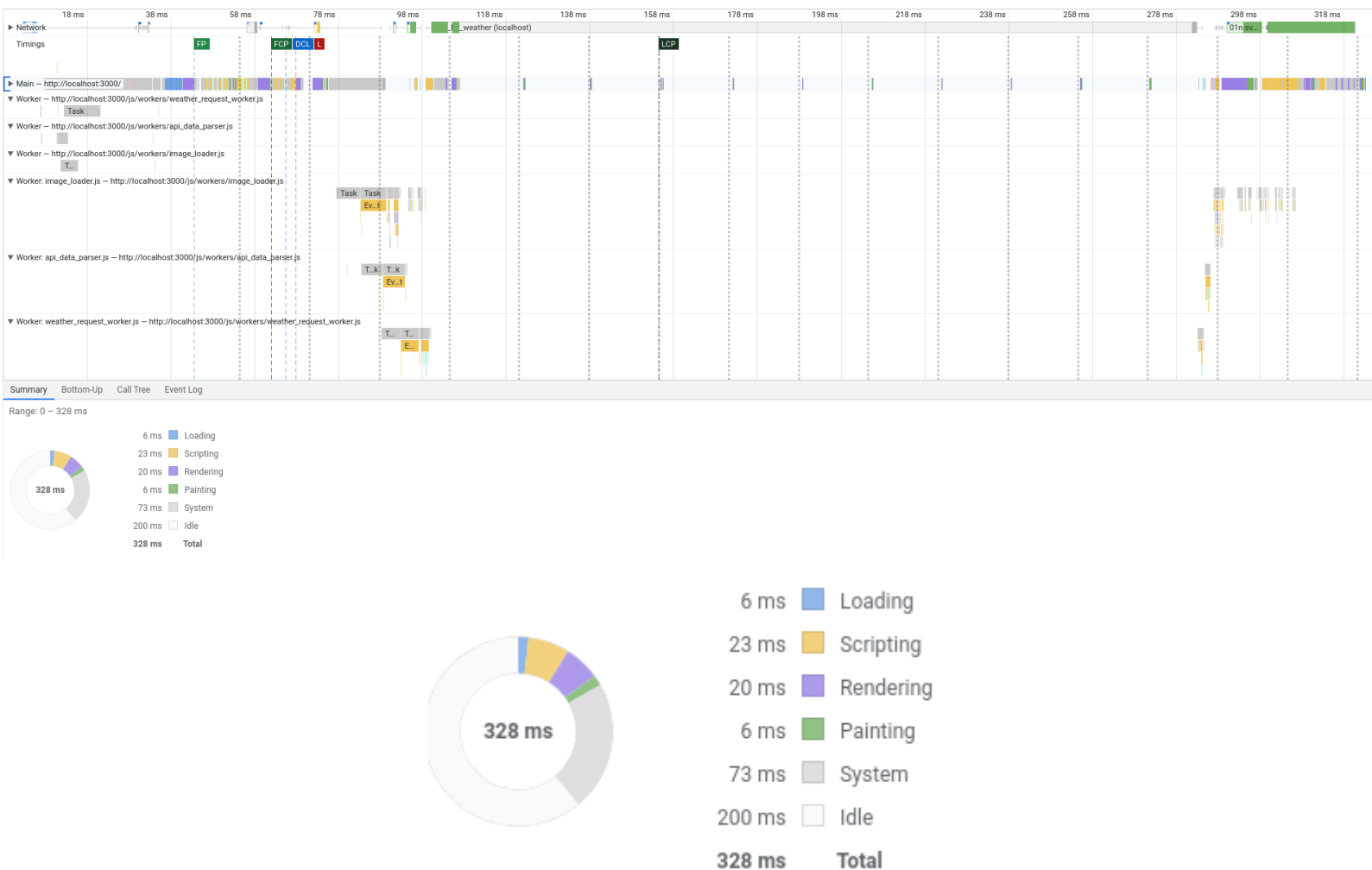


Mobile



Il Largest Contentfull paint è alto una delle cause è l'immagine di background che è inutilmente grossa per la versione mobile

Caricamento della pagina



5. Conclusioni

sono soddisfatto di come ha preso forma l'applicazione sotto tutti gli aspetti: grafica, performance e backhand, ma sicuramente alcune cose sono migliorabili

- l'immagine di background rappresentante lo skyline di New York è un' immagine molto grossa e per renderla di circa 55k ho dovuto sfuocarla e comprimerla nonostante ciò è ancora molto grossa ed anzi inutilmente grossa (in termini di larghezza) per la versione mobile. Sarebbe utile trovare un'altra soluzione ad esempio una foto più piccola o una foto più piccola solo per la versione mobile oppure ristrutturare la pagina e ridurre o eliminare lo spazio per l'immagine di background
- il servizio api utilizzato è gratuito ma ha un numero limitato di richieste: se gli utilizzatori della piattaforma diventano molti da saturare il numero di richieste

bisognerà apportare modifiche alla applicazione e/o cambiare servizio/abbonamento api

6. Nota sitografica

- il progetto su heroku: <https://cloudymaccarini.herokuapp.com/>
- bootstrap: <https://getbootstrap.com/>
- il link al repository: <https://github.com/LucaMaccarini/Cloudy>