



**Corso di Programmazione Web e Mobile**

**A.A. 2021-2022**

**MeteoCity**

**Luca Parenti - 909049**

Autore: Luca Parenti

Ultima modifica: 5 Settembre 2022 - Versione 2.0

Prima modifica: 26 Giugno 2022

# Relazione Progetto MeteoCity

Relazione dettagliata del progetto personale ‘MeteoCity’ per il corso di Programmazione Web e Mobile

## 1 Introduzione

MeteoCity è un’applicazione web che permette agli utenti la visualizzazione di informazioni metereologiche in tutto il mondo.

Lo scopo finale del progetto è quindi quello di fornire informazioni più o meno dettagliate all’utente riguardanti il meteo e le previsioni di pioggia (funzionalità disponibile solo per l’Italia).

### 1.1 Analisi dei requisiti

#### 1.1.1 Requisiti utente

Per l’utilizzo dell’applicazione all’utente non viene richiesto alcun tipo di esperienza (**non** deve conoscere *necessariamente* ciò che sta cercando), inoltre può essere utilizzato un qualsiasi dispositivo desktop o mobile.

L’utente può visualizzare la temperatura e la posizione rilevata dalla geolocalizzazione nella homepage, inoltre può visualizzare informazioni relative ad una specifica posizione sia cercando attivamente all’interno di una barra di ricerca, sia selezionando la posizione su una mappa.

All’utente è infine proposta una funzionalità di accesso/registrazione utile ai fini di salvataggio delle posizioni meteo preferite, in modo tale da poterle ritrovare in tutta facilità successivamente.

#### 1.1.2 Requisiti di sistema

- L’applicazione web implementa un modo di geolocalizzare l’utente e fornire dettagli basati sulla sua posizione.
- Questo è ritenuto un requisito fondamentale dal momento che l’applicazione perde quasi tutto il valore aggiunto rimuovendo i servizi di geolocalizzazione.

- L'applicazione web permette la ricerca e la visualizzazione di posizioni tramite una barra di ricerca ed una mappa visuale.
  - Questo requisito non richiede i permessi di geolocalizzazione.
- L'applicazione web fornisce un servizio di registrazione/login all'utente, per poi potergli permettere di avere accesso alle posizioni marcate come preferite.
- L'applicazione web fornisce un servizio di previsione pioggia nelle prossime ore con la visualizzazione radar satellitare fornita da 3Bmeteo ©.
  - **Attenzione** → questa feature è disponibile solo per la visualizzazione in Italia.

### 1.1.3 Requisiti di sviluppo

- 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 a un servizio NodeJS.
- Le chiamate devono interrogare o caricare dati in JSON o XML.

## 1.2 Destinatari

Agli utenti finali a cui si rivolge l'applicazione non viene espressamente richiesto alcun tipo di capacità puramente tecnica, viene richiesto però all'utente una minima capacità di navigazione e ricerca all'interno di un sito web.

L'applicazione è stata sviluppata con un design responsivo, quindi accessibile da tutti i dispositivi (desktop o mobile).

L'utente non necessariamente deve essere consapevole di ciò che sta cercando dato che nella homepage vengono già mostrate informazioni

riguardanti la sua posizione geolocalizzata (con possibilità di avere ulteriori dettagli con un singolo click), però ad utenti più esperti vengono anche forniti strumenti ulteriori, come la visualizzazione e la ricerca della posizione tramite una mappa.

### 1.3 Modello di valore

L'applicazione è stata ideata per un utilizzo molto semplice da parte dell'utente con contenuti ed informazioni minimali, ma *essenziali*.

Un valore aggiunto di questa applicazione è quello di poter salvare per ogni utente un elenco di posizioni preferite in modo tale da poterle ritrovare facilmente nella sua area personale.

Un altro contenuto che dà valore all'applicazione è quello di poter visualizzare un semplice radar meteo per previsioni di pioggia nel breve termine basato sulla geolocalizzazione (funzione disponibile solo per l'Italia).

### 1.4 Flusso dei dati

I dati necessari a ricevere informazioni metereologiche sono ottenuti dall'API resa disponibile gratuitamente da OpenWeatherMap e restituisce dati così strutturati:

```
{
  "coord": {
    "lon": 10.99,
    "lat": 44.34
  },
  "weather": [
    {
      "id": 501,
      "main": "Rain",
      "description": "moderate rain",
      "icon": "10d"
    }
  ],
  "base": "stations",
```

```
"main": {
  "temp": 298.48,
  "feels_like": 298.74,
  "temp_min": 297.56,
  "temp_max": 300.05,
  "pressure": 1015,
  "humidity": 64,
  "sea_level": 1015,
  "grnd_level": 933
},
"visibility": 10000,
"wind": {
  "speed": 0.62,
  "deg": 349,
  "gust": 1.18
},
"rain": {
  "1h": 3.16
},
"clouds": {
  "all": 100
},
"dt": 1661870592,
"sys": {
  "type": 2,
  "id": 2075663,
  "country": "IT",
  "sunrise": 1661834187,
  "sunset": 1661882248
},
"timezone": 7200,
"id": 3163858,
"name": "Zocca",
"cod": 200
}
```

I dati per il ‘radar satellitare’ invece sono link a GIF aggiornate di ora in ora sul sito di 3B Meteo, fornite direttamente dalla protezione civile.

Gli utenti all’interno del DB hanno la seguente struttura:

```
{
  _id: ObjectId("63131cd70d0d071f5efb4267"),
  email: 'test@test.com',
  password: '7iaw3Ur350mqGo7jwQrpkj9hiYB3Lkc/iBml1JQ0DbJ6wYX4o0',
  favourites: [ 'gerenzano', 'como', 'milano', 'limbiate' ]
}
```

Dove **favourites** è l’array che contiene le posizioni preferite per ogni utente creato.

I contenuti non sono pensati per la condivisione, ma solo per la consultazione personale del singolo utente, dal fatto che l’applicazione è stata ideata per essere utile personalmente all’utente nel caso in cui voglia tenere monitorate informazioni metereologiche in aree da lui specificate.

## 1.5 Aspetti tecnologici

### 1.5.1 Frontend

Per effettuare chiamate ed utilizzare le API necessarie all’applicazione viene utilizzato Fetch API di JavaScript, questo permette di ricevere i dati dal backend ed aggiornare le pagine in base ai dati richiesti.

Per la mappa interattiva è stato scelto di utilizzare la libreria open-source ‘leafletJS’ che permette la visualizzazione grafica della posizione (scelta o acquisita) dell’utente e l’interazione con l’utente per selezionare visualmente una posizione.

Viene utilizzata anche l'API di geolocation di HTML per ottenere la posizione dell'utente e mostrargli contenuti basati appunto sulla sua posizione geolocalizzata.

Nell'utilizzo di questa API è stato scelto di utilizzare il metodo `getCurrentPosition()` anzichè `watchPosition()` perchè la posizione sia presa una volta precisamente per info dettagliate sulla specifica posizione dell'utente anzichè aggiornata in base al suo movimento, questo perchè sia utile sia su dispositivi desktop che mobile.

Sono state inoltre implementate due chiamate AJAX all'interno del file `cities.js` che permettono l'update asincrono dell'array delle città preferite e dei bottoni di *aggiunta* e *rimozione* senza dover ricaricare la pagina.

Infine viene utilizzato Bootstrap 5 per facilitare la gestione e la costruzione dell'interfaccia utente e renderla piacevole alla vista, ma soprattutto minimale ed essenziale in modo tale che tutte le informazioni siano reperite in maniera semplice e veloce.

### 1.5.2 Backend

Il backend dell'applicazione si basa su Express.js ed ha il ruolo di fornire le pagine dell'applicazione richieste dall'utente tramite API ed inoltre di gestire API per la comunicazione sia con il server che con il database (funzionalità account utente e gestione informazioni meteo).

Come database è stato scelto MongoDB un famoso *document database* che sfrutta il formato JSON per l'interrogazione dei dati ed a cui si fa accesso tramite API REST.

È stato scelto questo tipo di DB per semplicità di sviluppo dato che MongoDB permette l'aggiornamento di interi documenti con operazioni elementari.

Infine il DB contiene una singola collezione **users** che gestisce gli utenti e per ogni utente un array di città preferite come specificato sopra.

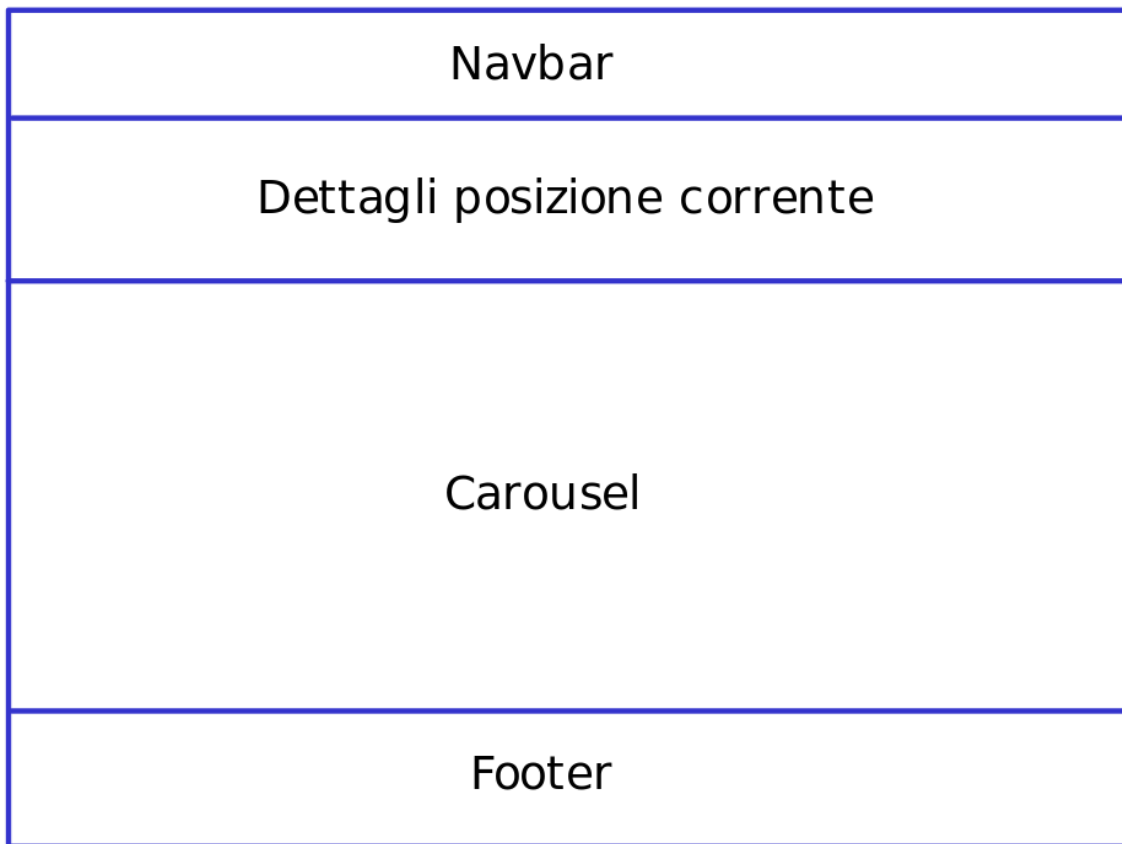
## 2 Interfacce

In tutte le pagine sono contenuti:

- Una barra di navigazione per muoversi tra le varie pagine del sito
- Uno switch per cambiare da *light mode* a *dark mode* e viceversa
- I pulsanti di *login* e *registrazione* se l'utente non ha eseguito l'accesso, oppure il pulsante di *logout* se lo ha eseguito

### 2.1 Homepage

Struttura della pagina:

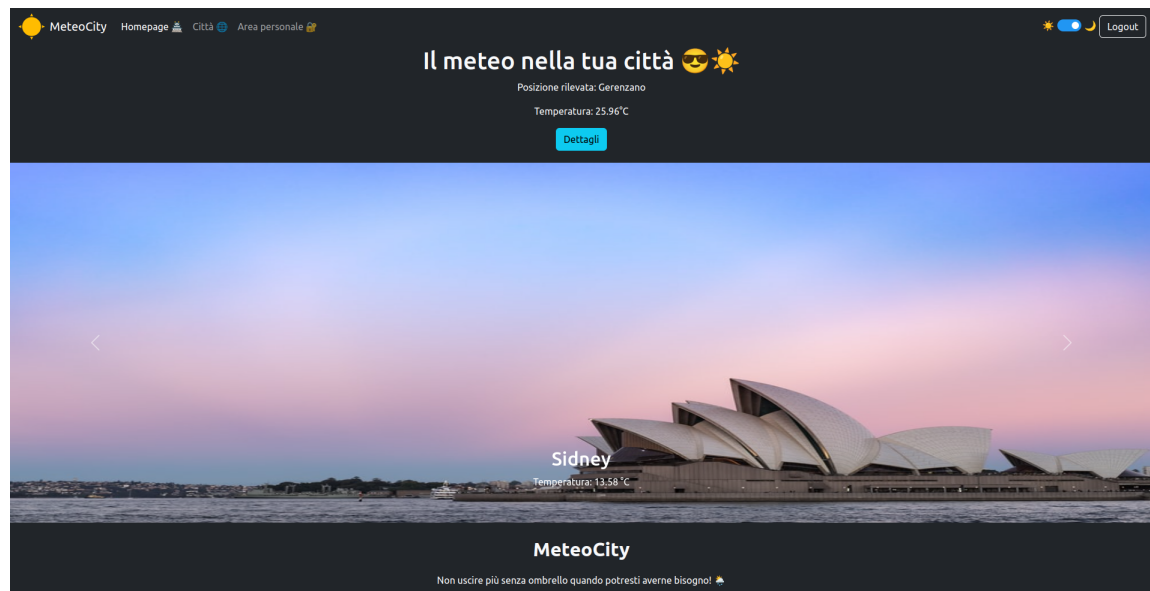


Questa pagina permette di visualizzare informazioni iniziali su temperatura relative alla posizione corrente. Inoltre cliccando sul pulsante **Dettagli** si verrà indirizzati alla pagina **cities** con maggiori informazioni.

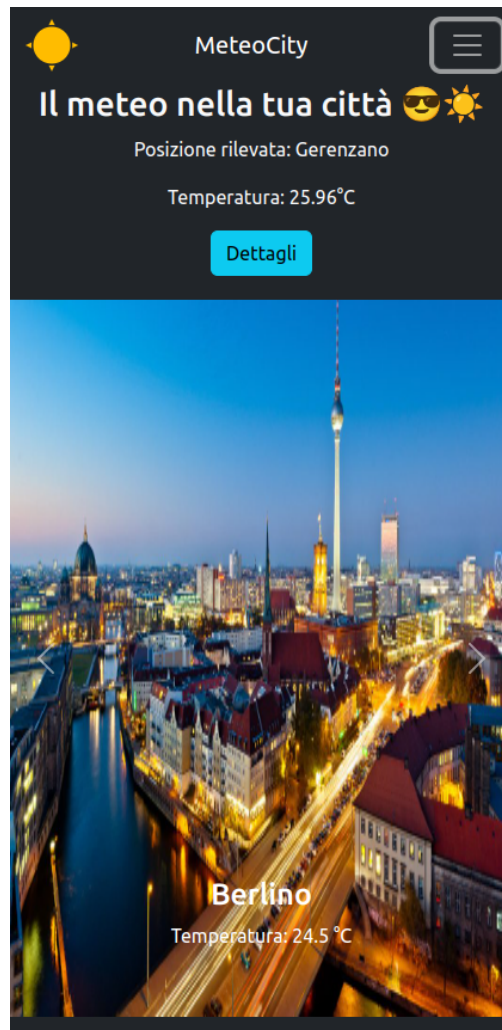
È infine presente un carousel che visualizza informazioni riguardanti 3 città predefinite: Milano, Sidney e Berlino.



## Interfaccia finale desktop:

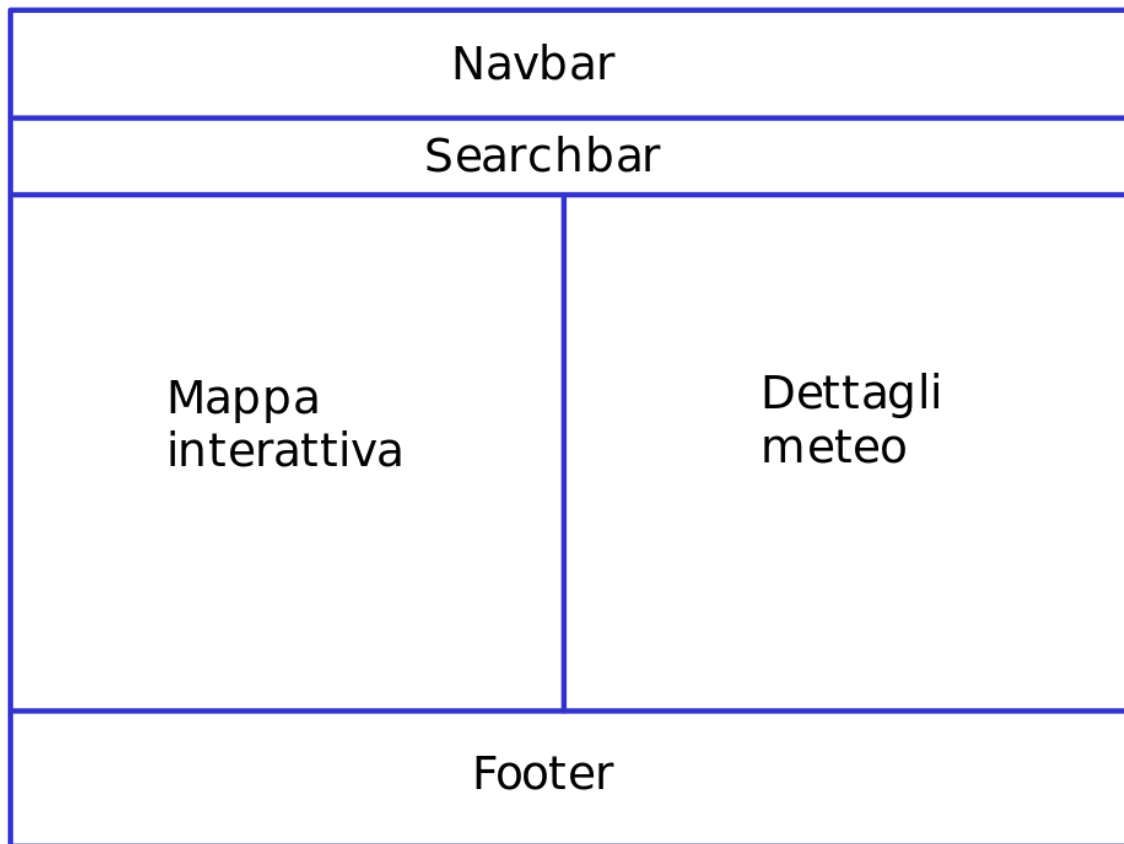


## Interfaccia finale mobile:



## 2.2 Cities

Struttura della pagina:



In questa pagina è possibile visualizzare graficamente sulla mappa interattiva la posizione dell'utente e cercarla tramite la searchbar posta appena sotto la navbar.

Nella parte sinistra della pagina (in modalità mobile appena sotto la searchbar) si può trovare la mappa interattiva su cui l'utente può cliccare per aggiornare la posizione e con essa anche le informazioni meteo. Informazioni meteo che si trovano sul lato destro della pagina (sotto la mappa in modalità mobile).

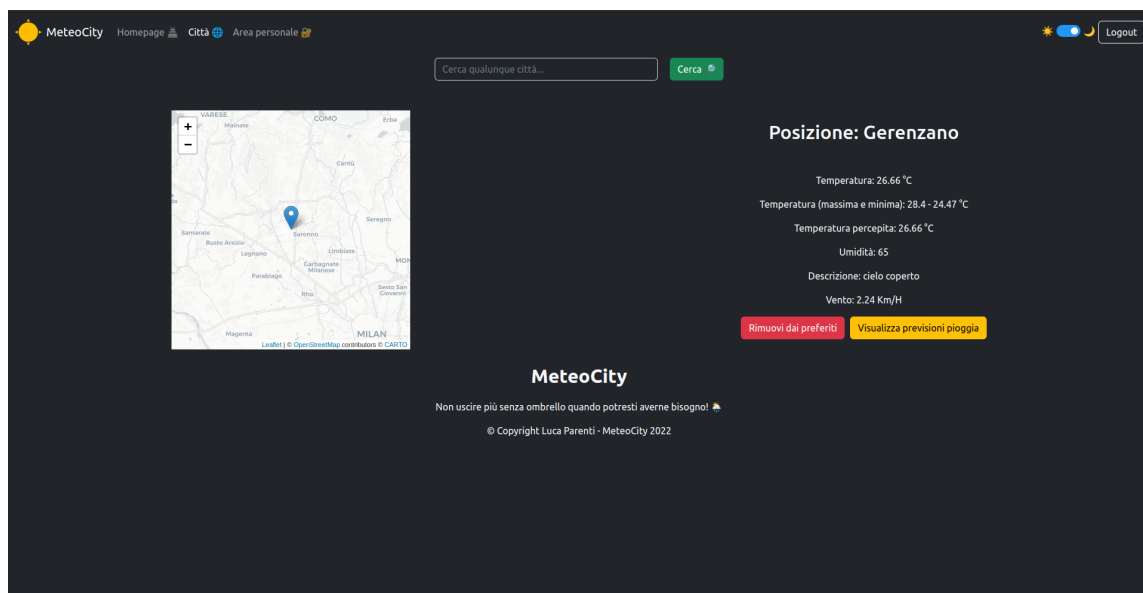
Sono disponibili infine due pulsanti:

- Aggiungi/Rimuovi dai preferiti → permette di aggiungere o rimuovere la posizione nelle posizioni preferite dall'utente.

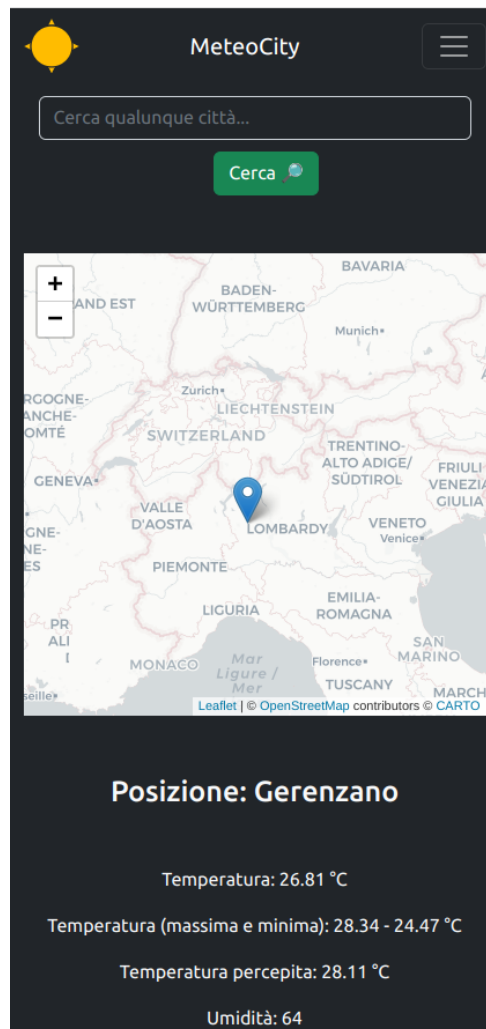
**Attenzione:** questo pulsante è disponibile e visibile solo ad un utente registrato.

- Visualizza previsioni pioggia → permette di aprire in un'altra pagina il radar meteo.

## Interfaccia finale desktop:

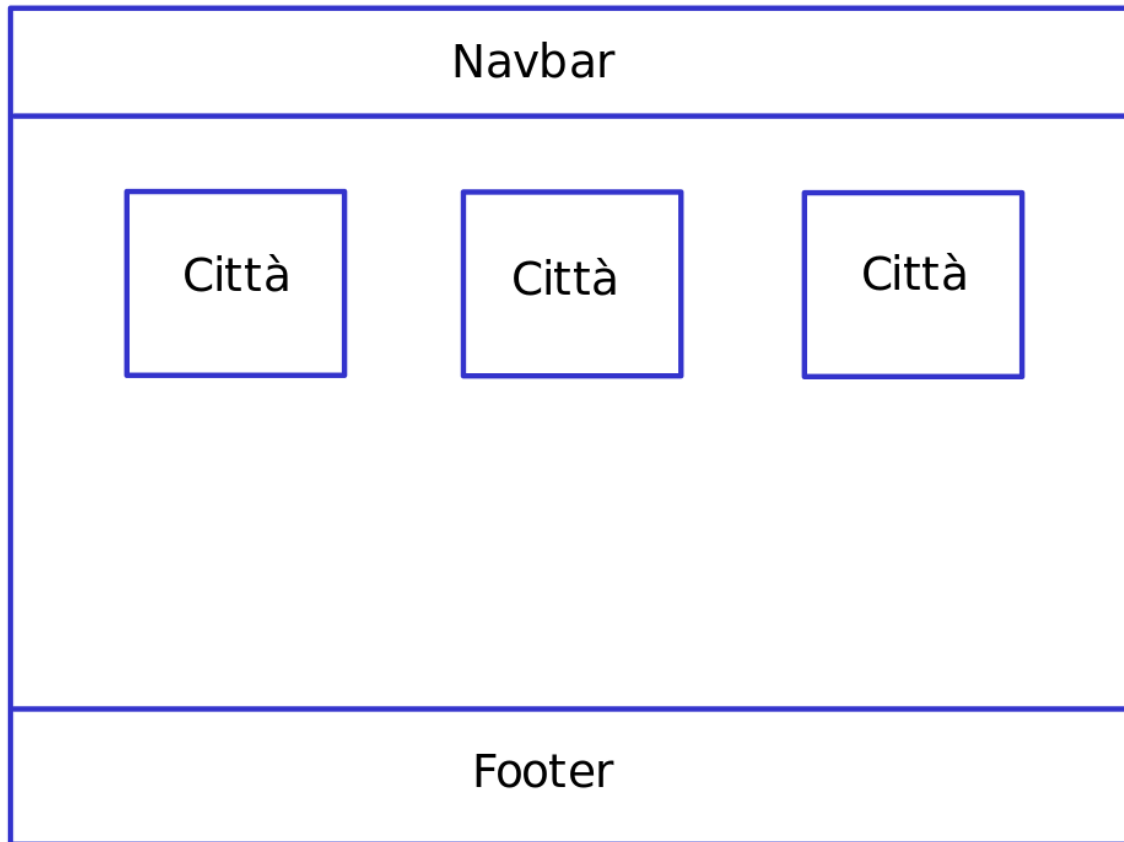


## Interfaccia finale mobile:



## 2.3 Dashboard

Struttura della pagina:

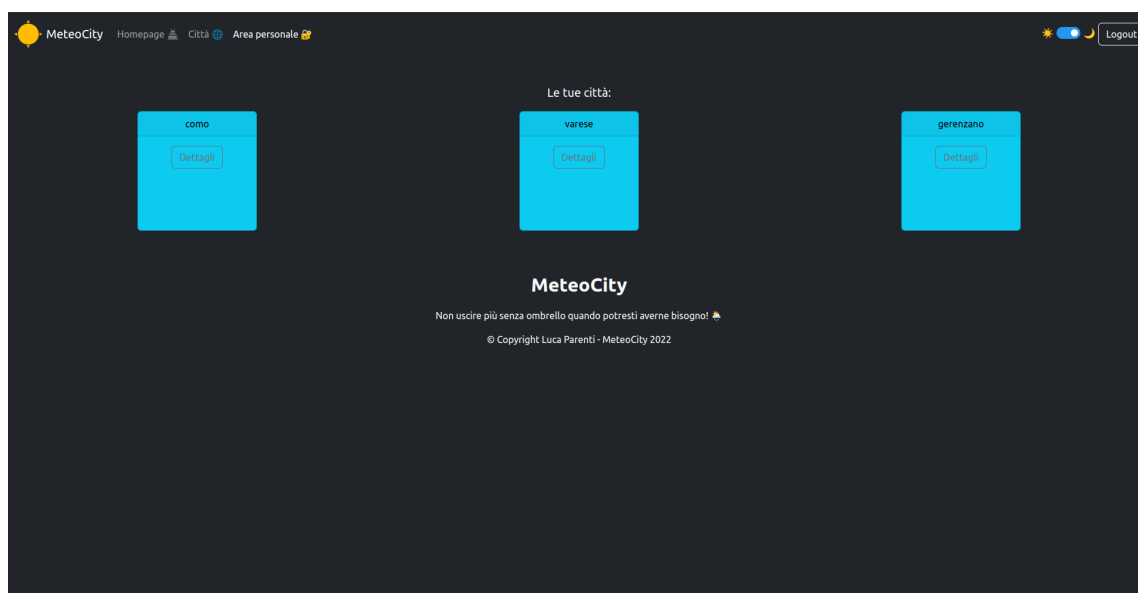


La dashboard permette all'utente di visualizzare le città da lui marcate come preferite.

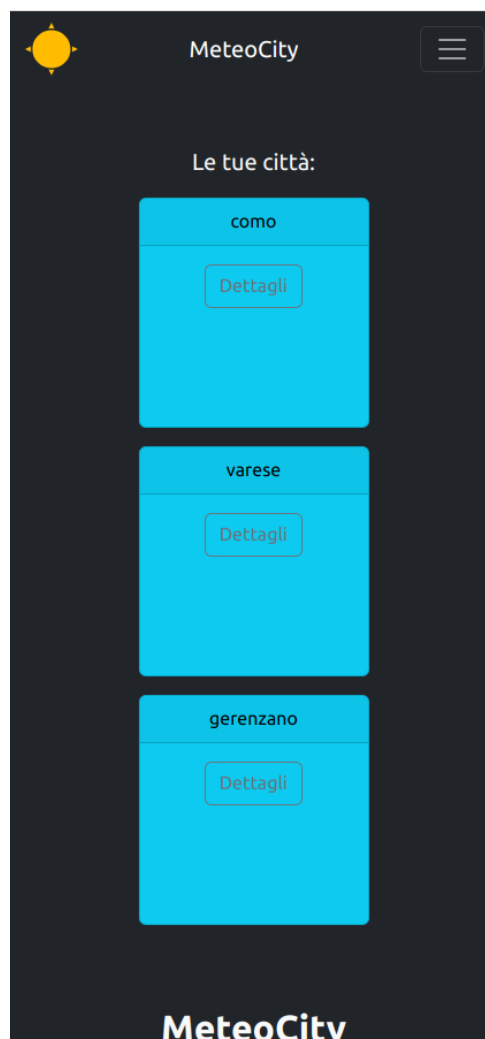
Per ogni città inoltre è disponibile un pulsante **Dettagli** che, reindirizzando l'utente alla pagina `cities`, permette di visualizzare informazioni meteorologiche come descritte al punto precedente.

**Attenzione:** se l'utente non è registrato verrà reindirizzato al login senza possibilità di accedere alla dashboard.

## Interfaccia finale desktop:

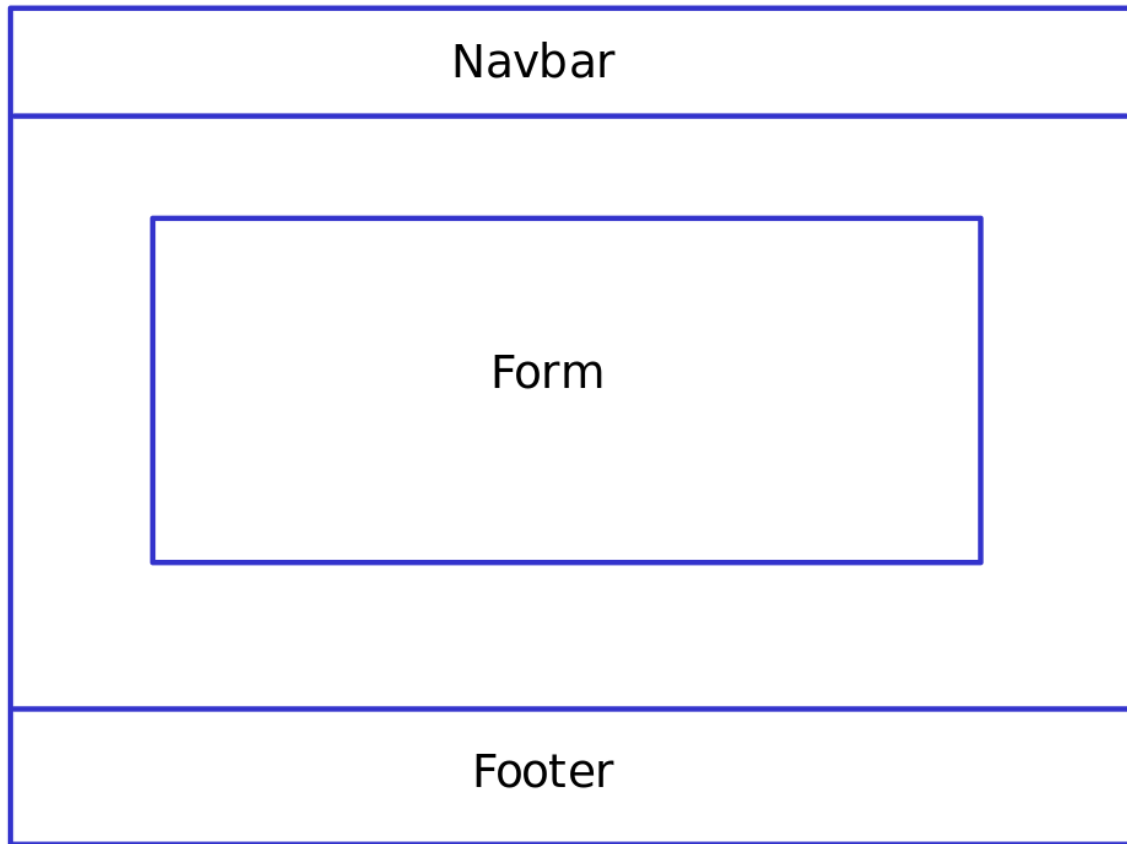


## Interfaccia finale mobile:



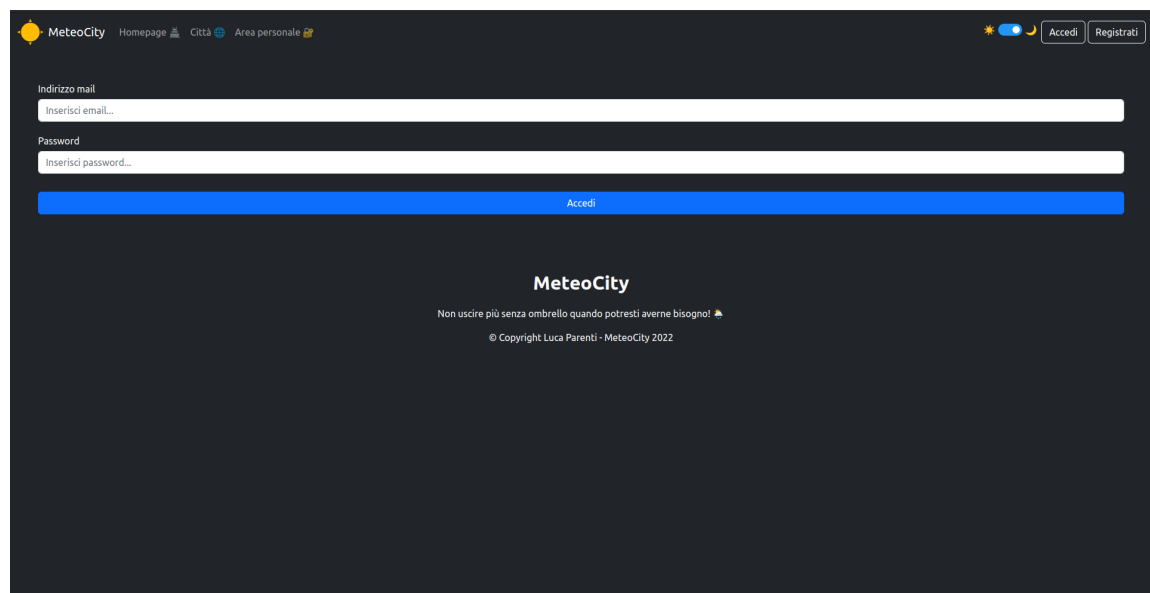
## 2.4 Login

Struttura della pagina:



Il login ovviamente permette all'utente (se registrato) di effettuare login nell'applicazione, attivando così la funzionalità della dashboard.

## Interfaccia finale desktop:



The desktop interface features a dark blue header with the MeteoCity logo and navigation links: Homepage, Città, and Area personale. On the right, there are icons for light/dark mode and buttons for 'Accedi' and 'Registrati'. The main content area contains a login form with labels 'Indirizzo mail' and 'Password', followed by input fields 'Inserisci email...' and 'Inserisci password...'. A prominent blue 'Accedi' button is positioned below the password field. At the bottom, the MeteoCity logo is displayed above a motivational quote and a copyright notice.

MeteoCity

Homepage Città Area personale

Indirizzo mail

Inserisci email...

Password

Inserisci password...

Accedi

**MeteoCity**

Non uscire più senza ombrello quando potresti averne bisogno! ☔

© Copyright Luca Parenti - MeteoCity 2022

## Interfaccia finale mobile:



The mobile interface is a vertical layout with a dark blue background. It includes the MeteoCity logo and a hamburger menu icon at the top. The login form consists of labels 'Indirizzo mail' and 'Password', followed by input fields 'Inserisci email...' and 'Inserisci password...'. A blue 'Accedi' button is located below the password field. The footer section contains the MeteoCity logo, a motivational quote, and a copyright notice.

MeteoCity

Indirizzo mail

Inserisci email...

Password

Inserisci password...

Accedi

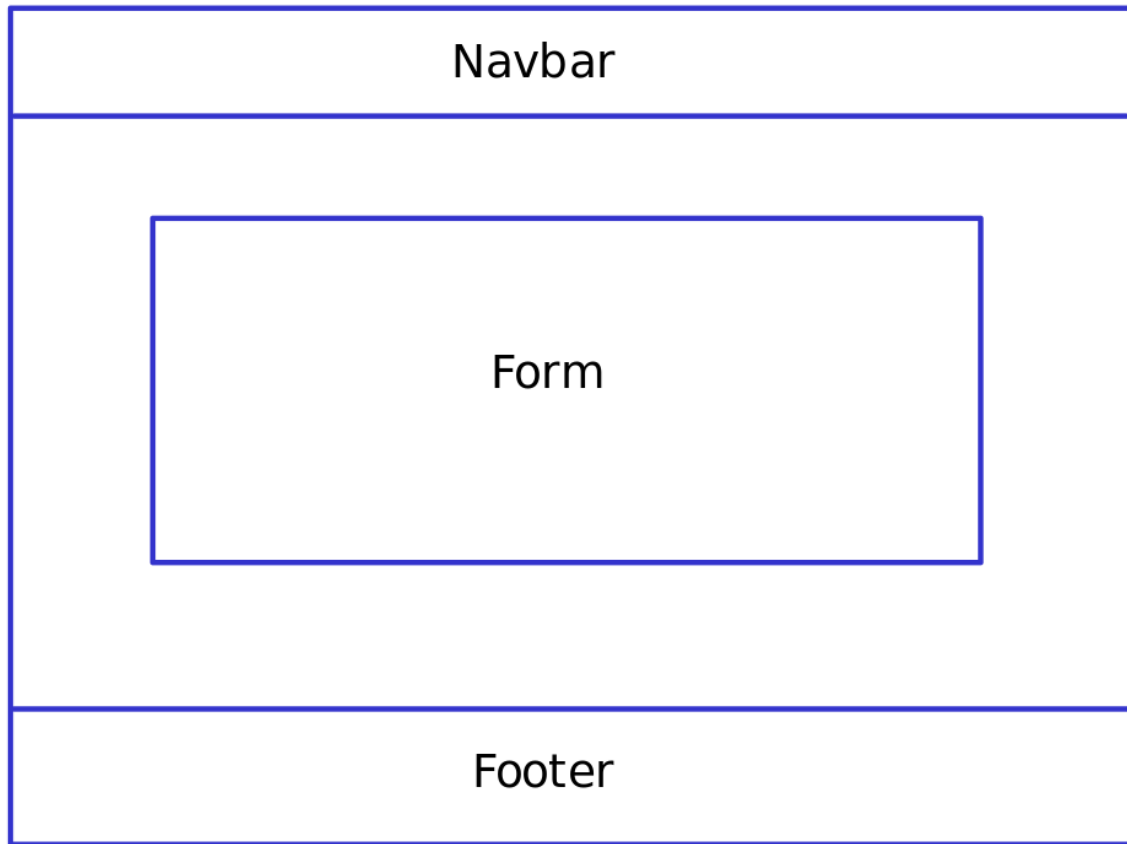
**MeteoCity**

Non uscire più senza ombrello quando potresti averne bisogno! ☔

© Copyright Luca Parenti - MeteoCity 2022

## 2.5 Register

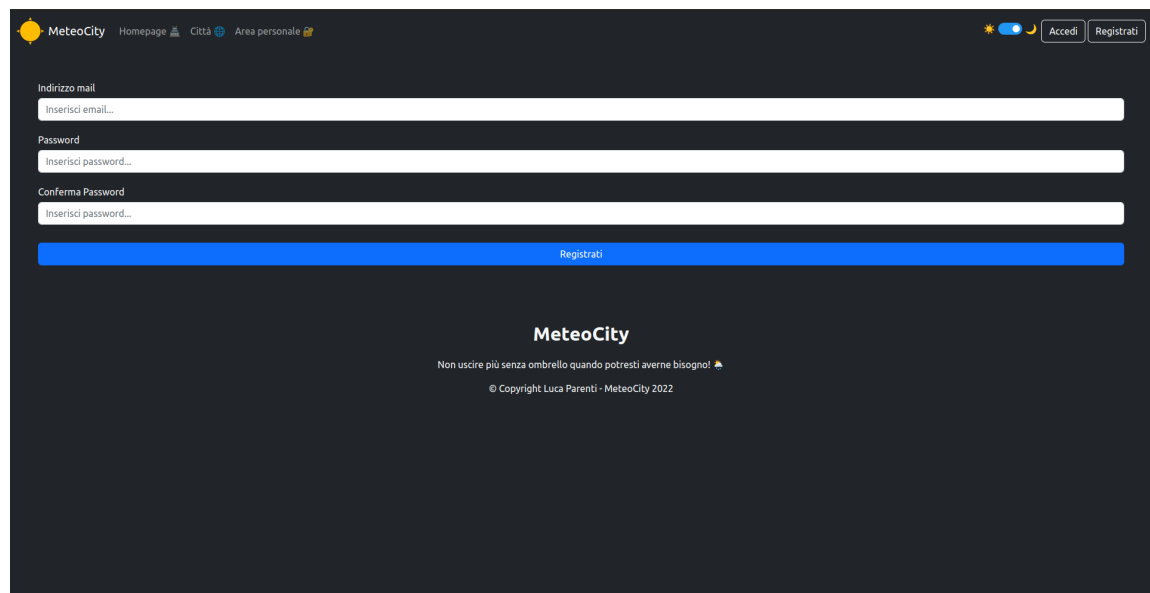
Struttura della pagina:



La pagina di registrazione ovviamente permette all'utente di effettuare la registrazione all'applicazione, attivando così la funzionalità della dashboard.

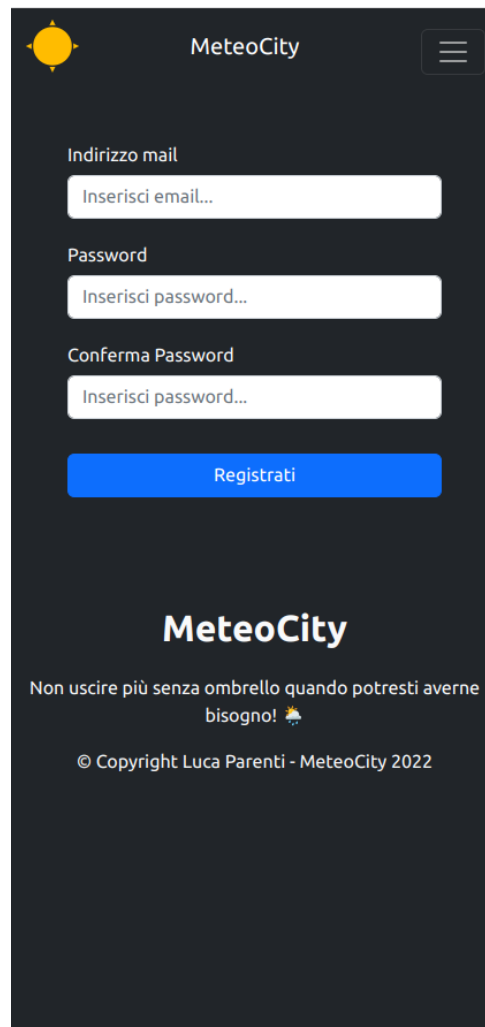


## Interfaccia finale desktop:



The desktop interface features a dark theme. At the top, a navigation bar includes the MeteoCity logo, links for 'Homepage', 'Città', and 'Area personale', a theme toggle, and 'Accedi' and 'Registrati' buttons. The registration form consists of three input fields: 'Indirizzo mail' (with placeholder 'Inserisci email...'), 'Password' (with placeholder 'Inserisci password...'), and 'Conferma Password' (with placeholder 'Inserisci password...'). A prominent blue 'Registrati' button is positioned below the fields. The footer contains the MeteoCity logo, a motivational quote 'Non uscire più senza ombrello quando potresti averne bisogno!', and the copyright notice '© Copyright Luca Parenti - MeteoCity 2022'.

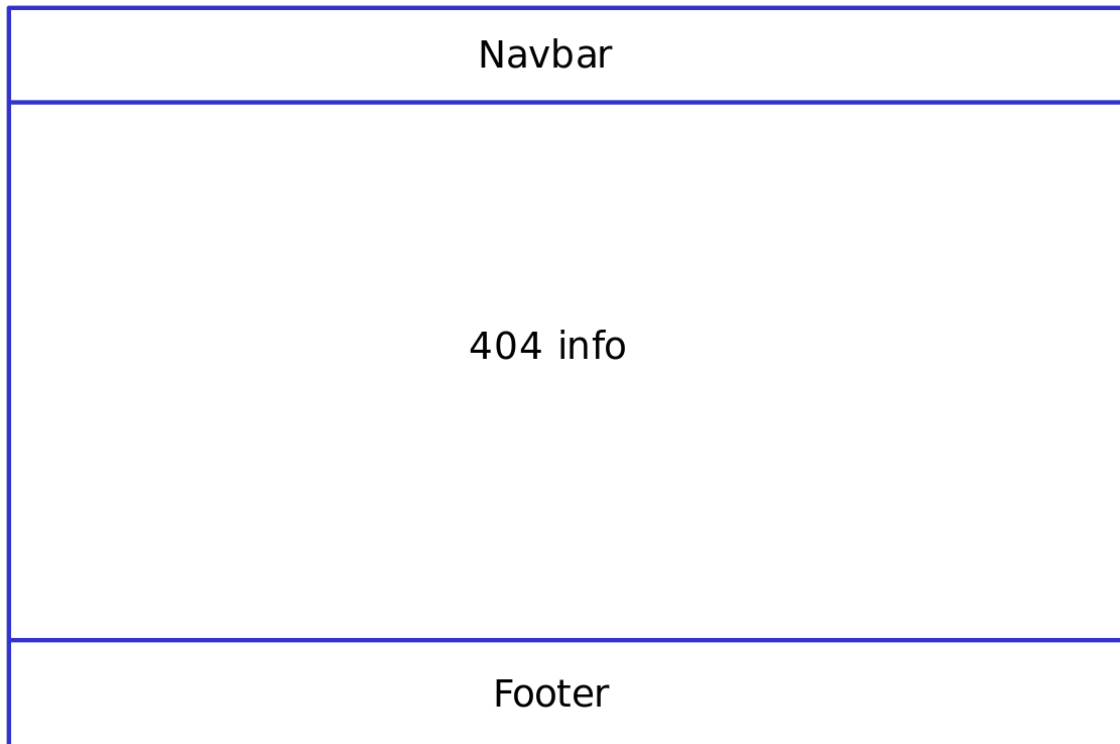
## Interfaccia finale mobile:



The mobile interface is designed for a vertical screen. It features a top bar with the MeteoCity logo, a hamburger menu icon, and the text 'MeteoCity'. The registration form is centered and includes three input fields: 'Indirizzo mail' (placeholder 'Inserisci email...'), 'Password' (placeholder 'Inserisci password...'), and 'Conferma Password' (placeholder 'Inserisci password...'). A blue 'Registrati' button is located below the fields. The footer displays the MeteoCity logo, the quote 'Non uscire più senza ombrello quando potresti averne bisogno!', and the copyright notice '© Copyright Luca Parenti - MeteoCity 2022'.

## 2.6 404

Struttura della pagina:



La pagina 404 'entra in gioco' quando l'utente inserisce nell'URI una risorsa non valida, di conseguenza viene mostrata questa pagina di errore.

## Interfaccia finale desktop:

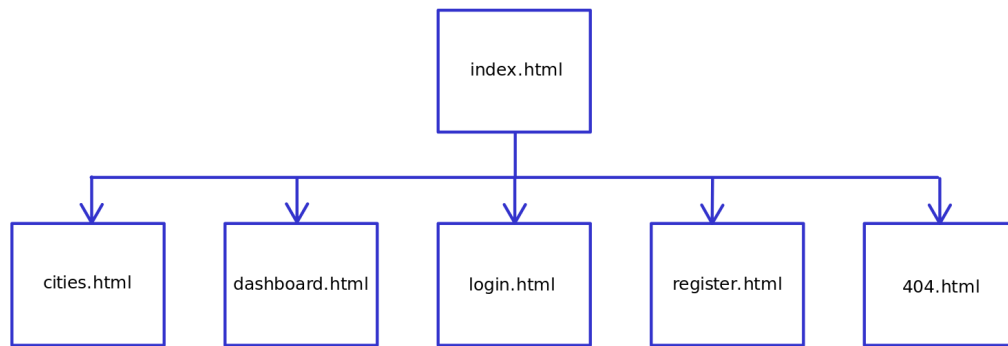


## Interfaccia finale mobile:

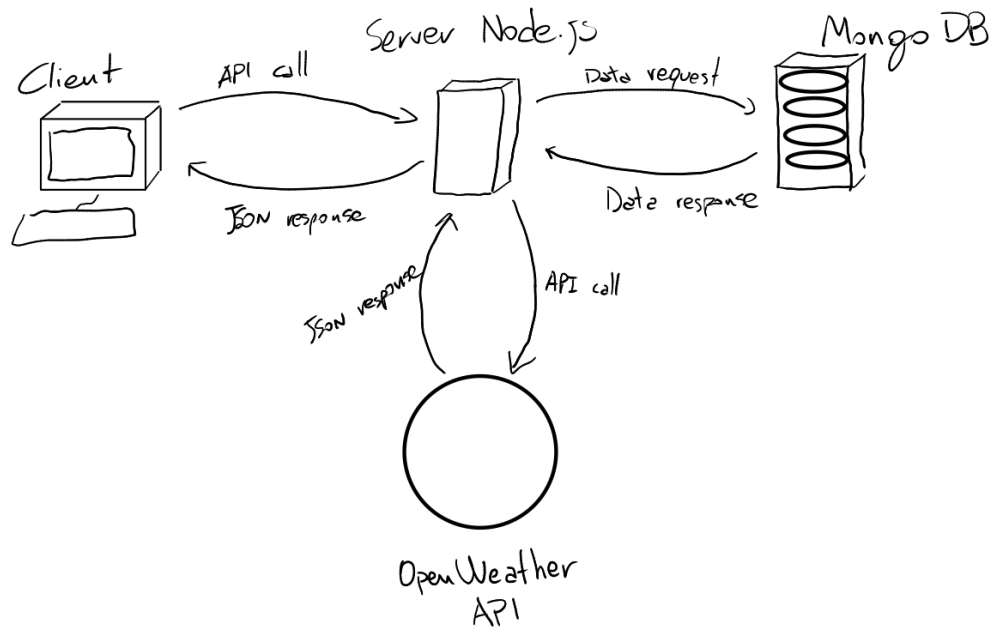


### 3 Architettura

#### 3.1 Diagramma ordine gerarchico



#### 3.2 Diagramma interazioni



## 4 Codice

### 4.1 app.js

Questo file è il più importante perchè si occupa di far partire il server Node.js oltre ad indirizzare e renderizzare le diverse views HTML e gestire le chiamate API sia in arrivo dal client, che in uscita, interfacciandosi con OpenWeather, ma anche con MongoDB per la gestione degli utenti.

```
src > JS app.js > ...
You, 2 ore fa | 1 author (You)
// All requires
1  const { hashPwd } = require('./public/js/helpers/pwd-hasher');
2  const User = require('./public/js/models/User');
3  const cookieParser = require('cookie-parser');
4  const session = require('express-session');
5  const bodyParser = require('body-parser');
6  const mongoose = require('mongoose');
7  const express = require('express');
8  const axios = require('axios');
9  const path = require('path');
10 require('dotenv/config');
11
12
13 // Tell the app to use express
14 const app = express();
15
16 // Every time we get a request we make sure that's correctly interpreted
17 app.use(bodyParser.urlencoded({ extended: true }));
18
19 // Tell app to use cookie parser
20 app.use(cookieParser());
21 app.use(express.json());
22
23 app.use(session({
24   secret: process.env.SESSION_KEY,
25   resave: false,
26   saveUninitialized: false,
27   // secure: true require HTTPS wich we don't have since we're on localhost, maxAge in milliseconds
28   cookie: {secure: false, maxAge: 3600000}
29 }));
30
31 // Tell express to look for the static views in public folder & what engine to use
32 app.set('views', path.join(__dirname, 'views'));
33 app.set('view engine', 'ejs');
34 app.engine('.html', require('ejs').renderFile);
35
36 // Tell express to look for the static css files in styles folder
37 app.use(express.static(path.join(__dirname, 'public')));
```

## Sezione rotte e connessione al DB:

```
src > JS app.js > ...
39 // SECTION - Routes
40 app.get('/', (req, res) => {
41   res.render('index.html');
42 });
43
44 app.get('/cities', (req, res) => {
45   res.render('cities.html');
46 });
47
48 app.get('/register', (req, res) => {
49   res.render('register.html');
50 });
51
52 app.get('/login', (req, res) => {
53   res.render('login.html');
54 });
55
56 app.get('/dashboard', async (req, res) => {
57   if (req.session.email !== undefined) {
58     const response = await User.findOne({
59       email: req.session.email
60     });
61
62     res.render('dashboard', {favourites: response.favourites});
63   } else {
64     res.render('login.html');
65   }
66 });
67
68 // SECTION - DB connection
69 mongoose.connect(
70   process.env.MONGODB_CONN,
71   { useNewUrlParser: true },
72   () => { console.log('Connected to DB!') }
73 );
```

## Sezione API OpenWeather:

```
src > JS app.js > ...
75 // SECTION - Weather APIs
76 app.get('/api/v1/getCityWeather/:city', (req, res) => {
77   const city = req.params.city;
78
79   axios.get(`${process.env.WEATHER_API_LINK}?q=${city}&appid=${process.env.WEATHER_API_KEY}&lang=it`)
80     .then((cityData) => axios.get(`${process.env.WEATHER_API_LINK}?lat=${cityData.data.coord.lat}&lon=${cityData.data.coord.lon}&appid=${process.env.WEATHER_API_KEY}&units=metric&lang=it`)
81       .then((weather) => res.send(weather.data))
82       .catch((err) => res.json({error: 'Città non trovata, Info: ${err}'})));
83 });
84
85 app.get('/api/v1/getPositionalWeather/:lat/:lon', (req, res) => {
86   const lat = req.params.lat;
87   const lon = req.params.lon;
88
89   axios.get(`${process.env.WEATHER_API_LINK}?lat=${lat}&lon=${lon}&appid=${process.env.WEATHER_API_KEY}&units=metric&lang=it`)
90     .then((weather) => res.send(weather.data))
91     .catch((err) => res.json({error: 'Città non trovata, Info: ${err}'})));
92 });
```

## Sezione API utente ed inizializzazione server:

```
src > JS app.js > ...
94 // SECTION - User APIs
95 app.post('/api/v1/users/signup', async (req, res) => {
96   const user = new User({
97     email: req.body.email,
98     password: hashPwd(req.body.password),
99     favourites: req.body.favourites
100   });
101
102   const confirm = hashPwd(req.body.confirmPassword);
103
104   if (user.password === confirm) {
105     await user.save()
106       .then(data => {
107         res.status(200);
108         res.json(data);
109       })
110       .catch(err => {
111         res.statusMessage = 'Errore in fase di creazione utente';
112         res.status(422);
113         res.json({error: `Errore, Info: ${err}`});
114       })
115   } else {
116     res.status(400);
117     res.json({error: 'Le password non coincidono!'});
118   }
119 });
120
121 app.post('/api/v1/users/login', async (req, res) => {
122   await User.findOne({
123     email: req.body.email,
124     password: hashPwd(req.body.password)
125   }).then(data => {
126     req.session.email = req.body.email;
127     res.status(200);
128     res.json(data);
129   }).catch(err => {
130     res.status(400);
131     res.send({error: `Info: ${err}`})
132   });
133 }
```

```
src > JS app.js > ...
135 app.get('/api/v1/users/logout', async (req, res) => {
136   if (req.session.email !== undefined) {
137     req.session.destroy();
138   }
139 });
140
141 // SECTION - favourites city
142 app.post('/api/v1/users/get-favourites', async (req, res) => {
143   await User.findOne({
144     email: req.body.email
145   }).then(data => {
146     res.status(200);
147     res.json(data.favourites);
148   }).catch(err => {
149     res.status(400);
150     res.send({error: `Info: ${err}`})
151   });
152 });
153
154 app.post('/api/v1/users/update-favourite', async (req, res) => {
155   const email = req.body.email;
156   const cityName = req.body.cityName;
157   var favourites = req.body.favourites;
158
159   if (cityName !== '') {
160     favourites.push(cityName);
161   }
162
163   await User.updateOne(
164     { email: email },
165     { $set: { favourites: favourites } }
166   ).then(data => {
167     res.status(200);
168     res.json({data: data, status: 200})
169   }).catch(err => {
170     res.status(422);
171     res.send({error: `Info: ${err}`})
172   });
173 }
```

```
175 // Catch all routes not defined to show custom 404 page
176 app.use((req, res, next) => {
177   res.status(404);
178   res.render('404.html');
179 });
180
181 // Start listening to port 3000
182 app.listen(3000);
```

## 4.2 cities.js - chiamate AJAX

Questa porzione di codice effettua chiamate AJAX all'API di gestione delle posizioni preferite dell'utente che si può vedere sopra nel codice di `app.js` per salvare/rimuovere posizioni preferite.

```
src > public > js > JS cities.js > showRainMap > navigator.geolocation.getCurrentPosition() callback
133
134 // SECTION - REMOVE - If we have to remove
135 if (mode === 'remove') {
136     const i = favourites.indexOf(cityName.toLowerCase());
137
138     // If index is found remove the item in that position
139     if (i > -1) {
140         favourites.splice(i, 1);
141     }
142
143     // AJAX call to update favourites array and buttons
144     xhttp.onreadystatechange = function () {
145         if (this.readyState === 4 && this.status === 200) {
146             removeButton.classList.add('hidden');
147             addButton.classList.remove('hidden');
148         }
149     };
150
151     xhttp.open("POST", "/api/v1/users/update-favourite");
152     xhttp.setRequestHeader("Content-type", "application/json");
153     xhttp.send(JSON.stringify({ email: email, cityName: '', favourites: favourites }));
154 }
155
156 // SECTION - ADD - If we have to add
157 if (mode === 'add') {
158     xhttp.onreadystatechange = function () {
159         if (this.readyState === 4 && this.status === 200) {
160             removeButton.classList.remove('hidden');
161             addButton.classList.add('hidden');
162         }
163     };
164
165     // AJAX call to update favourites array and buttons
166     xhttp.open("POST", "/api/v1/users/update-favourite");
167     xhttp.setRequestHeader("Content-type", "application/json");
168     xhttp.send(JSON.stringify({ email: email, cityName: cityName.toLowerCase(), favourites: favourites }));
169 }
170 }
```



## 4.3 cities.html

Questa porzione di codice rappresenta uno dei core dell'applicazione, se non la view fondamentale che permette maggiore interazione con l'utente rispetto alle altre.

```
src > views > cities.html > ...
78 <main>
79 <!-- SECTION - Search bar -->
80 <form action="/cities" method="GET">
81 <div class="form-group d-flex justify-content-center flex-sm-row flex-column">
82 <div class="centered-item">
83 <input
84   class="search form-control text-white bg-dark"
85   type="search"
86   placeholder="Cerca qualunque città..."
87   name="cityName"
88   required
89 >
90 </div>
91 <div class="centered-item">
92 <button type="submit" class="search-button btn btn-success">Cerca 🔍</button>
93 </div>
94 </div>
95 </form>
96
97 <!-- SECTION - Interactive -->
98 <div style="text-align: center;" class="d-flex justify-content-around flex-sm-row flex-column">
99 <!-- SECTION - Interactive Map -->
100 <div id="map" class="centered-item"></div>
101
102 <!-- SECTION - Forecast details -->
103 <div id="forecast" class="centered-item">
104 <h2 id="pos">Posizione: </h2>
105 <p id="temp">Temperatura: </p>
106 <p id="temp-maxmin">Temperatura (massima e minima): </p>
107 <p id="temp-perc">Temperatura percepita: </p>
108 <p id="hum">Umidità: </p>
109 <p id="descr">Descrizione: </p>
110 <p id="wind">Vento: </p>
111 <button onclick="handleFavourites('add')" id="button-favs-add" class="hidden btn btn-info">Aggiungi ai preferiti</button>
112 <button onclick="handleFavourites('remove')" id="button-favs-remove" class="hidden btn btn-danger">Rimuovi dai preferiti</button>
113 <button onclick="showRainMap()" id="satellite-rain-map" class="btn btn-warning">Visualizza previsioni pioggia</button>
114 </div>
115 </div>
116 </main>
```

## 5 Conclusioni

Questa applicazione non è stata pubblicata online, ma eseguita localmente.

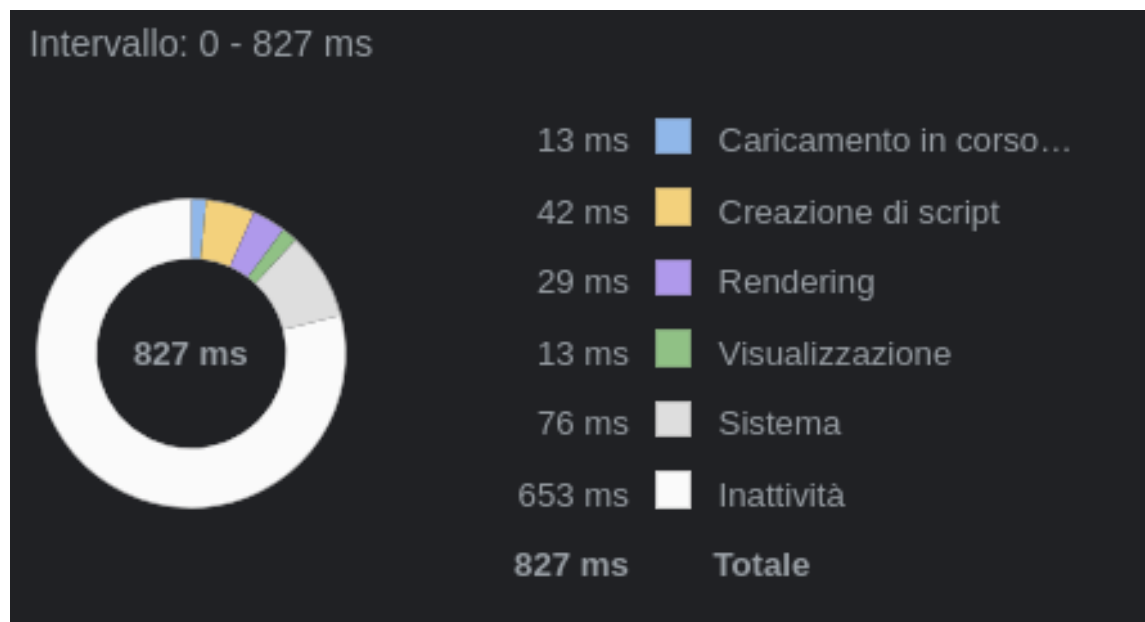
Tuttavia è stata ideata con l'intento di poter essere eseguita su qualsiasi macchina, per cui il codice con le istruzioni dettagliate è possibile trovarlo su GitHub al seguente link: <https://github.com/SuPaRuC/MeteoCity>.

Una difficoltà emersa durante lo sviluppo è che, per quanto riguarda il radar satellitare per le previsioni meteo, per l'Italia è reso disponibile da 3B Meteo solo per la zona Nord-Ovest o tutta l'Italia, di conseguenza se ci si trova in zona Nord-Ovest il radar sarà più specifico, altrimenti comprenderà tutta la nazione.

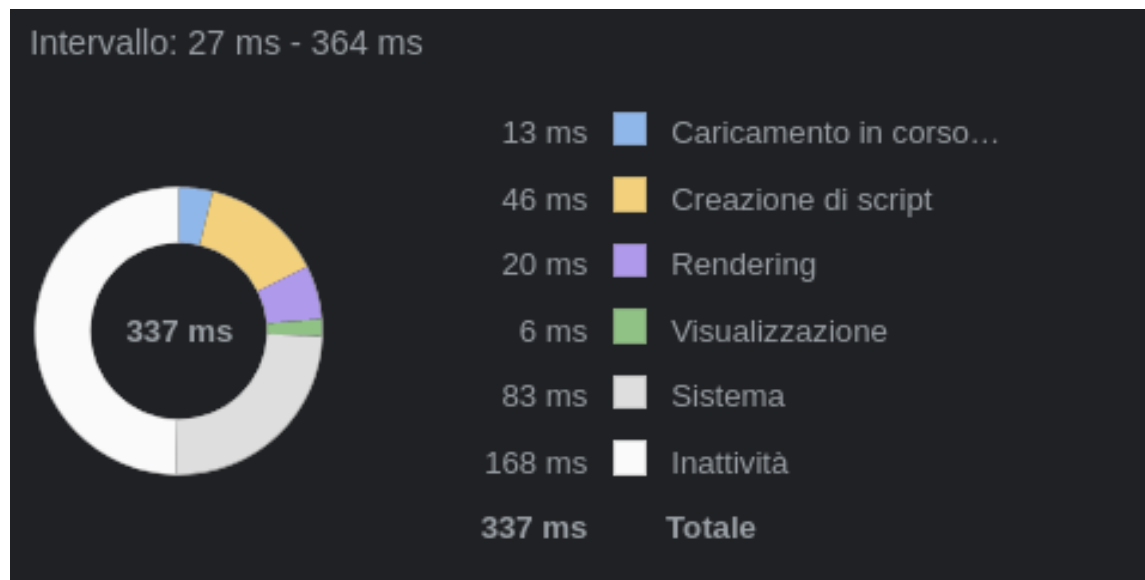
Come ulteriore sviluppo futuro si potrebbero aggiungere nuove regioni se non nazioni intere, cosa che, per mancanza di tempo, non è stato possibile approfondire.

Oltre a questo non ho incontrato altre particolari difficoltà durante lo sviluppo.

Durante lo sviluppo è stato eseguito un test di performance per vedere quanto sia stato effettivo l'utilizzo di un WebWorker per il caricamento delle immagini, il risultato è il seguente:



Senza WebWorker



Con WebWorker

Ora... Di certo l'incremento delle performance non è enorme, però c'è da considerare che le immagini caricate sono solamente 3, per cui se le immagini dovessero aumentare, di conseguenza l'incremento delle performance sarebbe quasi raddoppiato se non di più, considerando (come da immagini) i 6ms di differenza per la visualizzazione ed i quasi 10ms di differenza per il rendering. Non ci sono solo effetti positivi però, di contro avremmo un carico di sistema un pochino più pesante.

## 6 Nota bibliografica e sitografica

Risorse utilizzate per lo sviluppo:

- <https://www.w3schools.com/>
- <https://developer.mozilla.org/>
- <https://openweathermap.org/api>
- <https://www.mongodb.com/>