

# **Sviluppo di un'applicazione interattiva per l'esplorazione di aree verdi urbane**

Exploring urban green areas in an interactive web application

**Signoretti Loris**

Relazione finale di stage

**Relatore Schifanella Rossano**



Dipartimento di Informatica

Università degli Studi di Torino

2022

# Indice

<b>Indice</b>	<b>2</b>
<b>1 Dichiarazione di originalità</b>	<b>4</b>
<b>2 Introduzione</b>	<b>5</b>
<b>3 Frameworks e DBMS</b>	<b>6</b>
3.1 React . . . . .	6
3.2 Flask . . . . .	7
3.3 PostgreSQL . . . . .	7
3.3.1 PostGIS . . . . .	8
<b>4 Tecnologie e librerie</b>	<b>10</b>
4.1 JWT . . . . .	10
4.1.1 Struttura JWT Token . . . . .	12
4.2 Deck.gl . . . . .	12
4.2.1 DeckGL (componente React) . . . . .	13
4.2.2 Mapbox e react-map-gl . . . . .	13
4.2.3 GeoJSONLayer . . . . .	14
4.3 Ottimizzazioni . . . . .	15
4.4 H3 . . . . .	16
4.5 Axios . . . . .	17
<b>5 Server</b>	<b>18</b>
5.1 Blueprints . . . . .	19
5.1.1 Areas . . . . .	19
5.1.2 Auth . . . . .	19
5.1.3 Social . . . . .	19
5.1.4 Admin . . . . .	20
5.2 Librerie utilizzate . . . . .	21
5.2.1 Werkzeug . . . . .	21
5.2.2 Flask jwt extended . . . . .	21
5.2.3 SQLAlchemy . . . . .	21
5.2.4 GeoAlchemy2 . . . . .	21

5.2.5	Flask Cors . . . . .	22
5.2.6	H3 . . . . .	22
5.2.7	Shapely . . . . .	22
5.2.8	Validators . . . . .	22
5.3	Calcolo e popolazione nel database delle aree con pre computazione .	23
<b>6</b>	<b>Client</b>	<b>25</b>
6.1	Struttura ambiente di sviluppo e applicazioni React utilizzate . . . .	25
6.1.1	Apis . . . . .	25
6.1.2	Componenti . . . . .	25
6.1.3	Hooks . . . . .	26
6.1.4	Contesti . . . . .	27
6.2	Autenticazione . . . . .	29
6.3	Pagina principale . . . . .	32
6.3.1	Ricerca globale . . . . .	33
6.4	La mappa . . . . .	34
6.4.1	Campo di ricerca . . . . .	35
6.4.2	Pannello di dettaglio . . . . .	35
6.4.3	Mappa . . . . .	35
6.4.4	Redirect . . . . .	37
6.5	Social . . . . .	38
6.5.1	Tabs . . . . .	38
6.5.2	List . . . . .	38
6.5.3	Lazy Load e Infinite Scroll . . . . .	39
6.5.4	Pagine . . . . .	40
<b>7</b>	<b>Conclusioni</b>	<b>43</b>
7.1	Obbiettivi raggiunti . . . . .	43
7.2	Sviluppi futuri . . . . .	45

# 1 Dichiarazione di originalità

”Dichiaro di essere responsabile del contenuto dell’elaborato che presento al fine del conseguimento del titolo, di non avere plagiato in tutto o in parte il lavoro prodotto da altri e di aver citato le fonti originali in modo congruente alle normative vigenti in materia di plagio e di diritto d’autore. Sono inoltre consapevole che nel caso la mia dichiarazione risultasse mendace, potrei incorrere nelle sanzioni previste dalla legge e la mia ammissione alla prova finale potrebbe essere negata.”

## Riferimenti bibliografici

- [1] Java Brains. *How JSON Web Tokens Work*. URL: <https://medium.com/swlh/how-json-web-tokens-work-211ce7b705f7>.
- [2] Wikipedia. *Flask (informatica)*. URL: [https://it.wikipedia.org/wiki/Flask\\_\(informatica\)](https://it.wikipedia.org/wiki/Flask_(informatica)).
- [3] Wikipedia. *GeoJSON*. URL: <https://it.wikipedia.org/wiki/GeoJSON>.

## 2 Introduzione

”Green Areas” è un’applicazione web interattiva che permette l’esplorazione e la condivisione di aree verdi urbane presenti nelle maggiori città europee.

Si può vedere l’applicativo come diviso in due macro sezioni che vanno poi a collaborare: una è la parte inerente alla mappa, la sua ottimizzazione, il suo caricamento reattivo, e la sua interattività, l’altra è la parte social con tutto ciò che va a collegarsi alla mappa e l’esplorazione di aree verdi in base ai feed degli utenti e amici.

Pertanto, l’applicazione è stata sviluppata sia in ambito frontend con una web app dinamica sviluppata in React.js e un server in Flask che fornisce le api per consultare e modificare un database spaziale realizzato con PostgreSQL e PostGIS.

L’obiettivo principale è quello di poter usufruire di una mappa con caricamento delle aree in modo intelligente, poco alla volta mano a mano che ci si spostava in modo reattivo e senza delay grazie a chiamate poco pesanti per il server grazie a una precomputazione dei dati più pesanti mentre nella parte client di creare una grafica, sia accattivante provando a ”costruire” qualche piccola animazione ma anche semplice e ripetitiva nei punti simili in modo da poter riutilizzare i componenti e rendere più facile la loro modifica.

## 3 Frameworks e DBMS

Per la parte client("browser") ho utilizzato React e più nello specifico React.js; per la parte server ho invece utilizzato Python Flask e come database PostgreSQL.

Solitamente quando si realizza un progetto lato client in javascript si tende ad utilizzare javascript anche lato server in modo, in caso di necessità, di poter utilizzare librerie identiche e per maggiore praticità. Ho invece utilizzato python con il framework Flask in quanto, avendo utilizzato Python, oltre che per il progetto di sviluppo software anche per progetti miei personali, ero interessato ad imparare le sue caratteristiche in ambito server e fruizione di dati.

Inoltre Flask ha una buonissima integrazione con PostgreSQL e una buona facilità di utilizzo. Ho utilizzato PostgreSQL in quanto permette l'installazione sul database di estensioni che aggiungono alle nostre query funzionalità in più. Il mio interesse cadeva sull'estensione postGIS che implementa oggetti geografici per un database spaziale.

### 3.1 React

[React](#) è una libreria JavaScript per la creazione di interfacce utente interattive. Permette di progettare viste semplici per ogni stato dell'applicazione che React aggiornerà in modo efficiente e farà visualizzare i componenti giusti quando i dati cambiano. Le visualizzazioni dichiarative rendono il codice più prevedibile e più facile da eseguire per il debug.

È basato su componenti incapsulati che gestiscono il proprio stato, quindi componibili per creare interfacce utente complesse. Poiché la logica del componente è scritta in JavaScript anziché nei modelli, è possibile trasferire dati avanzati facilmente attraverso l'app e mantenere lo stato fuori dal DOM.

Non fa supposizioni sul resto dello stack tecnologico, quindi permette di sviluppare nuove funzionalità senza riscrivere il codice esistente. React può anche eseguire il rendering sul server utilizzando Node e alimentare app mobili utilizzando React Native.

I componenti React implementano un metodo `render()` che prende i dati di input

e restituisce cosa visualizzare. Questo esempio utilizza una sintassi simile a XML chiamata JSX. È possibile accedere ai dati di input passati al componente da render() tramite this.props.

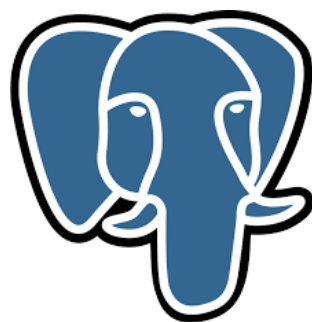
Oltre a prendere i dati di input (accessibili tramite this.props), un componente può mantenere i dati di stato interni (accessibili tramite this.state). Quando i dati di stato di un componente cambiano, il markup renderizzato verrà aggiornato richiamaudo nuovamente render().

## 3.2 Flask

Flask è un micro-framework Web basato su Python. È chiamato "micro-framework" perché ha un nucleo semplice ma estendibile. Non c'è uno strato di astrazione per la base di dati, validazione dei formulari, o qualsiasi altra componente per fornire funzionalità comuni per le quali esistono già librerie di terze parti.

Ad ogni modo, Flask supporta estensioni che possono aggiungere funzionalità a un'applicazione come se fossero implementate dallo stesso Flask. Ci sono per esempio estensioni per la validazione dei formulari, la gestione del caricamento dei file, varie tecnologie di autenticazione e altro.[2]

## 3.3 PostgreSQL



[PostgreSQL](#) è un sistema di gestione di database relazionali a oggetti (ORDBMS) basato su POSTGRES, sviluppato presso l'Università della California presso il Berkeley Computer Science Department. POSTGRES ha aperto la strada a molti concetti che sono diventati disponibili in alcuni sistemi di database commerciali solo molto più tardi.

PostgreSQL è un discendente open source di questo codice originale di Berkeley. Supporta gran parte dello standard SQL e offre molte funzionalità moderne:

- interrogazioni complesse
- chiavi esterne
- trigger
- viste aggiornabili
- integrità transazionale
- controllo della concorrenza multiversione

Inoltre, PostgreSQL può essere esteso dall'utente in molti modi, ad esempio aggiungendo nuovi

- tipi di dati
- funzioni
- operatori
- funzioni aggregate
- metodi di indice
- linguaggi procedurali

E grazie alla licenza libera, PostgreSQL può essere utilizzato, modificato e distribuito da chiunque gratuitamente per qualsiasi scopo, sia esso privato, commerciale o accademico.

### 3.3.1 PostGIS

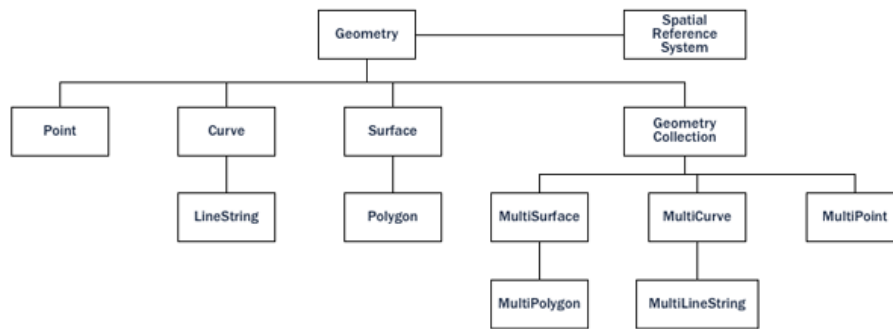




[PostGIS](#) è un'estensione per Postgres che permette di creare un database spaziale. I database spaziali archiviano e manipolano oggetti spaziali come qualsiasi altro oggetto nel database. Ci sono tre aspetti che associano i dati spaziali a un database: tipi di dati, indici e funzioni.

- I tipi di dati spaziali si riferiscono a forme come punto, linea e poligono;

#### Geometry Hierarchy



- L'indicizzazione spaziale multidimensionale viene utilizzata per l'elaborazione efficiente delle operazioni spaziali;
- Le funzioni spaziali, poste in SQL, servono per interrogare proprietà e relazioni spaziali.

La combinazione di questi tre permette di avere una struttura flessibile con performance ottimizzate. La creazione di un database spaziale è molto semplice e viene spiegata nel dettaglio nella [documentazione ufficiale](#).

## 4 Tecnologie e librerie

### 4.1 JWT

Per la sicurezza e il controllo dell'autorizzazione delle richieste da client a server ho utilizzato i token JWT che sta per [JSON Web Token](#), molto utilizzati per i piccoli servizi al giorno d'oggi.

Come spiegato in modo accurato da Java Brains[1], Http è un protocollo senza stato. Ciò significa che ogni interazione in Http deve contenere tutte le informazioni necessarie per tale interazione. Niente è ricordato da prima. Nessuno stato viene mantenuto su più richieste.

Quindi, quando hai un'applicazione server con pagine P1 e P2 accessibili solo da determinati utenti. Dici a un server su Http chi sei e di quale pagina hai bisogno. Il server ti autorizza ed eventualmente ti fornisce la pagina di cui hai bisogno ma se dovessi dare seguito a un'altra richiesta, il server non avrebbe idea di chi sei questa volta perché non "ricorda" quale sia la tua richiesta precedente. In ogni interazione, devi fornire tutti i dettagli e le informazioni richieste per tale interazione. Non esiste nessuna dipendenza da informazioni precedenti.

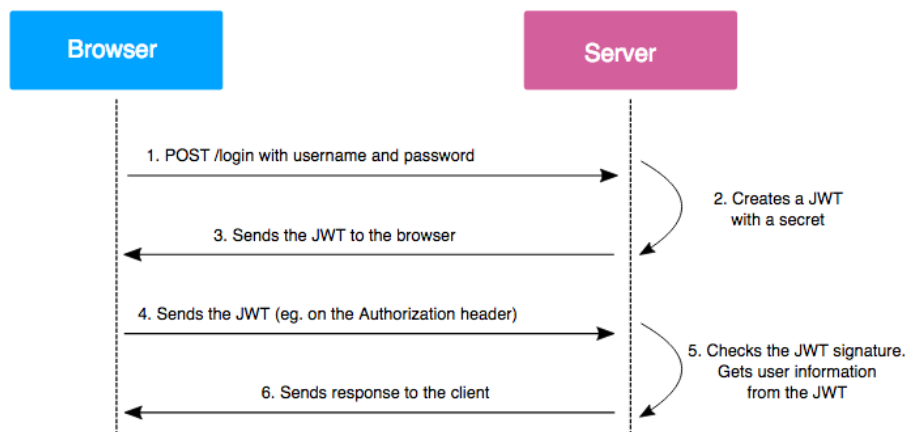
Questo andrebbe bene per richieste di pagine statiche ma nel nostro caso abbiamo bisogno di pagine dinamiche in base a chi siamo e a che ruolo abbiamo. Esistono diversi modi in cui le applicazioni Web gestiscono e ricordano le sessioni e due dei modi più diffusi utilizzano entrambi qualcosa chiamato token. Le due opzioni popolari sono:

1. Utilizzo dei token di sessione
2. Utilizzo di token Web JSON (JWT).

Quando un utente si autentica all'interno del sito lato server viene generata una nuova sessione in cui tiene traccia delle operazioni svolte e restituisce l'id di sessione al client. In questo modo ad ogni futura richiesta il client passerà anche l'id della sessione in modo che il server sappia precisamente chi sta facendo la richiesta. Il metodo per inviare l'id all'interno della richiesta dipende dall'implementazione ma il modo più utilizzato è quello di salvare questo id in un cookie in modo che sia automaticamente aggiunto nell'header di tutte le richieste verso il server.

Un problema di questo approccio è per esempio l' utilizzo di più server con un load balancer che decide di passare il traffico al server meno carico. Come possiamo capire se la mia richiesta di autenticazione viene effettuata sul server 1 e un' altra mia richiesta sul server 2, quest'ultimo non ha idea di chi io sia in quanto lo sa solamente il server su cui mi sono autenticato. Questo può essere risolto con una cache condivisa per le sessioni, tipico utilizzo di Redis, oppure convogliando tutte le richieste di un singolo client sempre allo stesso server.

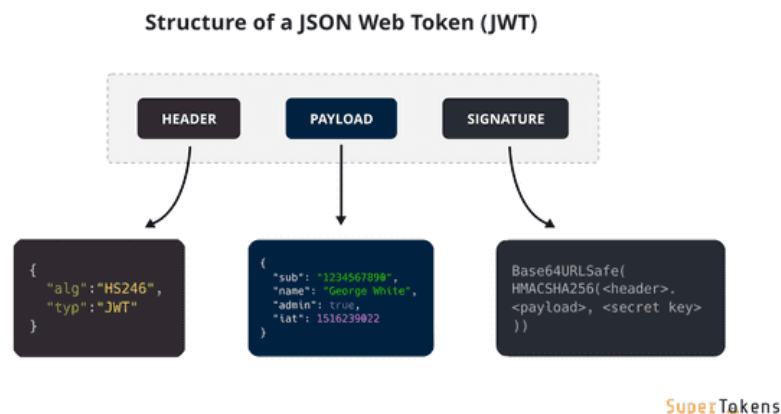
Con l' utilizzo dell token jwt invece, il server non deve tenere traccia delle informazioni dell' utente in sessione e passargli l' id di quest'ultimo come token, ma passa direttamente le informazioni dell' utente come token ovvero un oggetto json che contiene tutte le informazioni necessarie. Ovviamente quest' ultimo viene “firmato” in modo che quando viene passato al server in una futura transazione questo possa verificarne l'autenticità.



Possiamo riassumere la differenza tra token di sessione e jwt come il primo un token di riferimento e il secondo come un token di valore. Ovviamente anche i token jwt possono essere salvati nei cookie per automatizzare il passaggio nelle richieste.

Un JSON Web Token è un token di accesso standardizzato secondo [RFC 7519](#) e consente lo scambio sicuro di dati tra due parti. Contiene tutte le informazioni importanti su un'entità, in modo che non sia necessaria alcuna interrogazione del database e che la sessione non debba essere memorizzata sul server (stateless session). I JSON Web Token sono quindi particolarmente popolari nei processi di autenticazione. I brevi messaggi possono essere criptati e quindi fornire informazioni sicure su chi è il mittente e se lo stesso sia dotato dei permessi di accesso necessari.

### 4.1.1 Struttura JWT Token



**Header:** contiene il tipo di token e l'algoritmo utilizzato per la firma.

**Payload:** qui sono contenute le informazioni effettive dell'utente passate come copie key/value.

**Signature:** firma del token creata utilizzando la codifica dell'header e del payload con il metodo specificato e una chiave segreta. Da un lato, questa firma verifica che il messaggio non sia stato modificato durante il percorso e dall'altro, se il token è firmato con una chiave privata, assicura che il mittente del JWT sia quello corretto.

## 4.2 Deck.gl

[Deck.gl](#) è un framework basato su WebGL per l'analisi dei dati esplorativi visivi di grandi set di dati. Segue un approccio a più livelli per visualizzazione dei dati deck.gl e consente di ottenere rapidamente risultati visivi impressionanti con il minimo sforzo componendo livelli esistenti o sfruttando l'architettura estensibile di deck.gl per soddisfare esigenze personalizzate. Esegue calcoli ad alta precisione emulando calcoli in virgola mobile a 64 bit nella GPU, deck.gl renderizza i set di dati con precisione e prestazioni senza precedenti. Le API deck.gl sono progettate per riflettere il paradigma della programmazione reattiva. Sia che utilizzi Vanilla JS o React, è in grado di gestire un rendering WebGL efficiente con un carico di dati elevato. Sebbene deck.gl funzioni in modo autonomo senza una mappa di base, funziona bene con i fornitori di mappe di base preferiti come Google Maps, Mapbox, ArcGIS e altri. Laddove la libreria di mappe di base lo consenta, deck.gl può interlacciare con livelli di mappe 3D per creare visualizzazioni senza interruzioni.

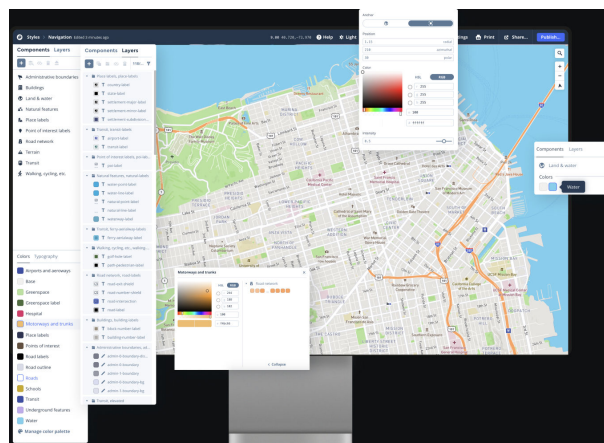
### 4.2.1 DeckGL (componente React)

DeckGL è l'interfaccia principale di deck.gl per le applicazioni React. DeckGL è un componente React che prende un elenco di istanze livello di deck.gl e uno stato di visualizzazione (view in prima persona, a mappa ecc.) e renderizza tali livelli su una tela trasparente che può essere utilizzata come sovrapposizione ad altri componenti come per esempio una mappa. La classe DeckGL è un wrapper React della classe Deck JavaScript e richiede quindi i medesimi prop(input).

La documentazione anche se l'ho trovata complicata su certi aspetti per la personalizzazione è abbastanza precisa e permette di comprendere il tutto anche grazie ad esempi e piccoli progetti sulla piattaforma GitHub.

### 4.2.2 Mapbox e react-map-gl

Mapbox è un'azienda che mette a disposizione dati e strumenti per la visualizzazione di mappe online. Mette a disposizione mappe con una vasta personalizzazione tramite Mapbox Studio di colori e oggetti che vogliamo far visualizzare come strade, fiumi eccetera, proprio come se stessimo modificando tramite Photoshop.



Permette di creare navigazioni fluide e interattive su percorsi aiutandoci a trovare la strada più veloce con una semplice ricerca.

Il mio utilizzo principale è stato quello di utilizzare la mappa Mapbox come base per Deck gl su cui andare a sovrapporre i diversi modelli. Per fare ciò Mapbox mette a disposizione Mapbox GL JS, una libreria javascript che permette di utilizzare mappe vettoriali sul web con buone prestazioni e styling in tempo reale. Ho utiliz-

zato però react-map-gl che è un wrapper di questa libreria per React che fornisce le stesse funzionalità con l'utilizzo di componenti reattivi personalizzabili.

### 4.2.3 GeoJSONLayer

La libreria di deck.gl mette a disposizione diversi layer per la visualizzazione dei dati per esempio per inserire linee, punti, aree circolari e come nel nostro caso poligoni. GeoJSON è un formato aperto utilizzato per archiviare una collezione di geometrie spaziali i cui attributi sono descritti attraverso JavaScript Object Notation. Le geometrie possibili sono punti (come ad esempio indirizzi o toponimi), linee spezzate (come percorsi, strade e confini), poligoni (paesi, province, laghi), e collezioni multiple di queste tipologie. Le geometrie GeoJSON non devono però necessariamente rappresentare entità geografiche: ad esempio, i software di navigazione assistita possono usarlo per descrivere l'area di copertura del servizio.<sup>[3]</sup>

Esempio:

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [11.1215698, 46.0677293]
      },
      "properties": {
        "name": "Fontana dell'Aquila"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [11.1214686, 46.0677385], [11.121466, 46.0677511], [11.1213806, 46.0681452],
```

```

        [11.1213548,46.0682642],[11.1213115,46.0684385],[11.1212897,46.0685261],
        [11.1212678,46.0686443]
    ]
},
"properties": {
    "lanes": 1,
    "name": "Via Rodolfo Belenzani"
}
}
]
}

```

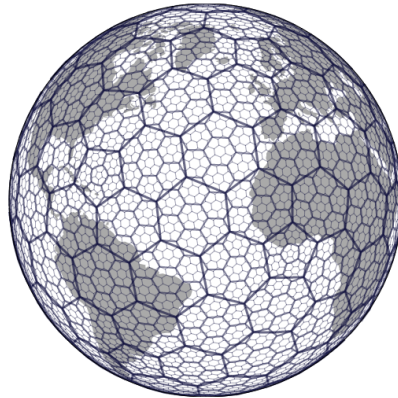
Utilizzando questo formato per inviare dati dal server al browser possiamo andare ad inserirli come dati del nostro layer GeoJSON e farli visualizzare sulla mappa, ovviamente il layer consente di modificare diversi parametri tra cui colori, spessore e eventi per click e hover.

## 4.3 Ottimizzazioni

La [documentazione](#) spiega anche come ottimizzare la propria applicazione, utile soprattutto per dispositivi meno performanti, come:

- Evitare modifiche inutili ai dati dei layer che richiederebbero render inutili
- Usare update Triggers, ovvero delle guardie che eseguono il render solamente quando necessario
- Non creare un solo layer con tutti i dati ma piuttosto caricarne poco alla volta in più layer
- Al posto di cancellare e rimettere layer preferire di renderli invisibili

## 4.4 H3

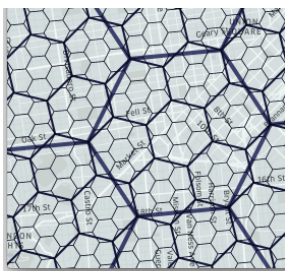


[H3](#) è un sistema di indicizzazione geospaziale open source realizzato da Uber che divide il mondo in celle esagonali. La libreria H3 Core implementa il sistema a griglia H3. Include funzioni per la conversione di coordinate (latitudine e longitudine) alla cella H3 in cui sono contenute, prendere il centro delle celle H3, prendere il confine delle celle H3, trovare i vicini delle celle H3 e altro ancora.

I sistemi a griglia sono fondamentali per analizzare grandi insiemi di dati spaziali, suddividendo le aree della Terra in celle di griglia identificabili.

Con questo in mente, Uber ha sviluppato H3, il sistema a rete per ottimizzare in modo efficiente i prezzi di corse e spedizioni, per visualizzare ed esplorare dati spaziali. H3 permette a Uber di analizzare le informazioni geografiche per stabilire prezzi dinamici e prendere altre decisioni a livello cittadino.

H3 fonde un sistema a griglia esagonale con un sistema gerarchico con indici a 64 bit: la risoluzione va da 0 a 15 dove la quindicesima è la più fine e precisa mentre la prima è quella più grande. Si può immaginare come tanti livelli sempre più precisi, ogni cella esagonale ha 7 figli ad una risoluzione maggiore e farà parte dei sette figli di una cella di risoluzione minore.



Gli esagoni non possono essere suddivisi perfettamente in sette esagoni, quindi le celle più fini sono contenute solo approssimativamente all'interno di una cella madre. Gli identificatori di queste celle figlie possono essere facilmente troncati per trovare la cella antenata con una risoluzione più grossolana, consentendo un'indicizzazione efficiente.



La libreria offre diversi metodi per l'esplorazione delle celle per esempio:

- trasformare coordinate in celle e il contrario
- prendere i figli di una cella
- prenderne la cella madre
- prendere le celle che ci circondano
- calcolare distanze tra celle
- calcolare perimetri di celle
- trovare i riempimenti di aree con celle di una data risoluzione

## 4.5 Axios

[Axios](#) è un client HTTP basato su Promise per node.js e il browser. È isomorfo (= può essere eseguito nel browser e nodejs con la stessa base di codice). Sul lato server utilizza il modulo http nativo node.js, mentre sul client (browser) utilizza XMLHttpRequests. Permette di:

- Creare XMLHttpRequests dal browser
- Effettua richieste http da node.js
- Utilizzare l'API Promise
- Intercettare richieste e risposte
- Trasformare i dati per richieste e risposte
- Cancellare le richieste
- Trasformare in automatico i dati JSON
- Avere supporto lato client per la protezione contro XSRF

## 5 Server

Server Struttura ambiente Come visto prima per l'implementazione del server ho utilizzato Flask che mette a disposizione tutto il necessario per creare un ambiente ordinato e facilmente modificabile.

Python permette di creare degli environment virtuali in modo da tenere installate le librerie per il progetto locali e non installate in tutto il sistema il che permette grazie ad alcuni comandi di creare un file di installazione delle librerie e d'altra parte di installarle tutte con un semplice comando.

Flask inoltre ha un proprio environment (`.flaskenv`) che ci permette di definire delle costanti globali che sono le impostazioni del nostro progetto per esempio la cartella dove si trova l'applicativo (`src`), il fatto di essere in un server in fase di development che permette di vedere log più specifici e nel nostro caso l'url di connessione del db, le risoluzioni per le nostre aree `h3` e la key per generare i JSON Web Tokens. Questi ultimi, però, in caso di deploy del server al pubblico dovrebbero poi essere spostati in un environment privato non accessibile dagli utenti in modo da mantenere il tutto segreto (`.env`).

Il vero applicativo Flask contenuto nella cartella `src` è diviso in diversi file per modularità, vi è un file principale che necessariamente per essere riconosciuto tale si chiama `__init__.py`.

Per essere reso modulare Flask mette a disposizione degli oggetti chiamati Blueprints, essi lavorano come un'applicazione Flask ma non lo sono, possono essere definiti dei moduli aggiuntivi che l'applicativo può registrare per aggiungere funzionalità, delle estensioni. Il nostro file principale si occupa di inizializzare l'applicativo, settare le impostazioni necessarie a database e jwt, registrare i blueprint e settare CORS.

Con tutto questo non creiamo delle api (Application Programming Interfaces) che ci permettono di fare richieste al nostro server attraverso fetch lato client.

## 5.1 Blueprints

Abbiamo quattro blueprint specifici, uno per la gestione delle aree verdi, uno per la parte social, uno per l'autenticazione e uno per l'amministratore. Alla propria creazione il Blueprint richiede un nome e un prefisso url che sarà la route dove verranno convogliate le richieste verso di esso. Abbiamo le seguenti route:

- /api/areas
- /api/auth
- /api/social
- /api/admin

### 5.1.1 Areas

Implementa tutte le route per la richiesta di aree verdi, aree h3 e i loro collegamenti:

- Prendere tutte le aree verdi
- Prendere le aree verdi in base all'indice h3
- Prendere gli indici h3
- Prendere le città

### 5.1.2 Auth

Implementa le route per l'autenticazione:

- Login
- Logout
- Refresh del token di accesso
- Registrazione

### 5.1.3 Social

Implementa le route per i like, i follow e la gestione del proprio account:

- Mettere like
- Togliere like
- Seguire
- Smettere di seguire
- Prendere like in base all'utente
- Prendere like di un'area verde
- Prendere i follow di un utente
- Prendere informazioni generali di un utente
- Prendere le aree in classifica
- Prendere utenti e aree in base a un valore di ricerca

#### **5.1.4 Admin**

Implementa le route per l'amministratore per la modifica dei dati:

- Creazione tabelle database
- Cancellazione tabelle database
- Inserire aree da GeoJSON
- Inserire aree da file inerente a una città
- Inserire città
- Cancellare città
- Cancellare aree verdi singole
- Prendere tutti gli utenti
- Creare un utente
- Generare dei dati fittizi (usato in fase development)

## 5.2 Librerie utilizzate

### 5.2.1 Werkzeug

Tradotto dal tedesco significa attrezzo, è infatti una libreria che offre molte utilità per Python. Ho utilizzato la sua sottoparte security che offre metodi per la generazione di password criptate e il loro controllo.

### 5.2.2 Flask jwt extended

Libreria per la gestione di token jwt con metodi per generare i token (access e refresh), prendere l'identità (il parametro) utilizzata per generare i token, nel nostro caso l'id, settare i token all'interno di cookie e rimuoverli.

Offre inoltre un decoratore per le route che permette di accedervi solamente se si ha un token, e un metodo che fa lo stesso ma al momento desiderato dell'esecuzione nella route, questi ci servono per bloccare le route che richiedono di essere autenticati.

### 5.2.3 SQLAlchemy

SQLAlchemy è il toolkit Python SQL e Object Relational Mapper che offre agli sviluppatori di applicazioni tutta la potenza e la flessibilità di SQL. SQLAlchemy fornisce una suite completa di noti modelli di persistenza a livello aziendale, progettati per un accesso al database efficiente e ad alte prestazioni, adattati in un linguaggio di dominio Pythonico semplice.

### 5.2.4 GeoAlchemy2

Utilizzo di SQLAlchemy con database spaziali. GeoAlchemy 2 fornisce estensioni a SQLAlchemy per lavorare con i database spaziali. GeoAlchemy 2 si concentra su PostGIS. Sono supportati PostGIS 1.5, PostGIS 2 e PostGIS 3. Anche SpatiaLite è supportata, ma l'utilizzo di GeoAlchemy 2 con SpatiaLite richiede una configurazione specifica lato applicazione. GeoAlchemy2 funziona con SpatiaLite 4.3.0 e versioni successive (tranne per gli alambicchi che richiedono SpatiaLite  $\geq 5$ ). GeoAlchemy2 mira ad essere più semplice del suo predecessore, GeoAlchemy. Più semplice da usare e più semplice da mantenere.

### 5.2.5 Flask Cors

Un'estensione Flask per la gestione della Cross Origin Resource Sharing (CORS), rendendo possibile AJAX multiorigine. Questo pacchetto ha una filosofia semplice: quando vuoi abilitare CORS, desideri abilitarlo per tutti i casi d'uso su un dominio. Ciò significa che non dovrai più perdere tempo con intestazioni, metodi e così via consentiti.

### 5.2.6 H3

Abbiamo visto precedentemente la libreria in cosa consiste. Viene utilizzata quindi per la richiesta di aree verdi al server dal client passando un indice di cella. Per avere chiamate reattive i dati vengono pre computati come vedremo dopo. Importante è il metodo `ST_As_GeoJSON` che utilizzato all'interno di una query ci restituisce un'area automaticamente in formato GeoJson che viene accettato dal layer Deckgl al cui interno ha un parametro `properties` contenente tutti i valori come nome centro e area del poligono.

### 5.2.7 Shapely

Shapely è un pacchetto Python con licenza BSD per la manipolazione e l'analisi di oggetti geometrici planari. Si basa sulle librerie GEOS (il motore di PostGIS) e JTS (da cui GEOS è stato portato) ampiamente distribuite. Shapely non si occupa di formati di dati o sistemi di coordinate, ma può essere facilmente integrato con pacchetti che lo sono.

### 5.2.8 Validators

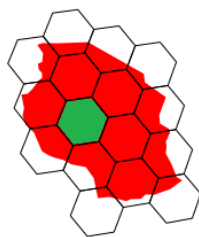
Python ha tutti i tipi di strumenti di convalida, ma ognuno di essi richiede la definizione di uno schema. Questa libreria offre strumenti di convalida senza richiedere la definizione di un modulo o di uno schema. L'ho utilizzata per controllare la forma di un'email.

### 5.3 Calcolo e popolazione nel database delle aree con pre computazione

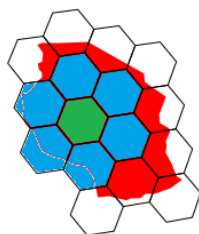
Per avere dati sempre pronti e non dover calcolare i dati da restituire al client ad ogni richiesta viene eseguita una pre computazione al caricamento delle aree verdi.

All'inserimento di una nuova area viene registrato nel database con il suo nome, la città in cui si trova e il suo poligono geometrico e il calcolo della sua area e centro. In seguito per ogni risoluzione H3 che abbiamo scelto in base ai test vengono calcolate le celle che interessano il poligono utilizzando metodi della libreria H3:

1. vengono mantenuti due array, uno per l'intersezione e uno per il bordo da controllare
2. viene trovata la cella a partire dalle coordinate centrali del poligono e viene aggiunta ad entrambi gli array

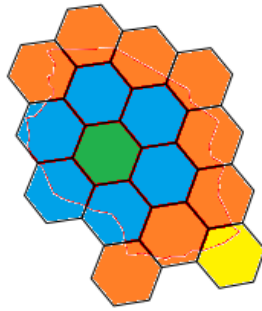


3. fino a quando l'array bordo non è vuoto:
  - a) prendo la cella che sta in testa all'array
  - b) ne calcolo l'anello di celle che la circonda



- c) per ognuno di questi vicini se non è all'interno dell'intersezione:
  - i. utilizzando la libreria Shapely controllo che la cella interseca geometricamente con il poligono iniziale

ii. se interseca la aggiungo all'intersezione e al bordo

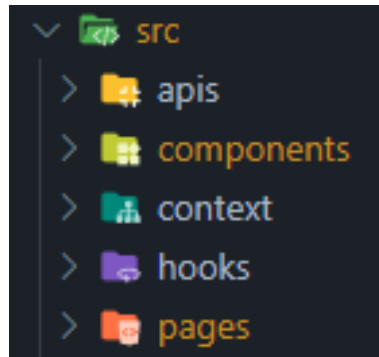


Nella tabella di relazione tra aree verdi e celle H3 viene aggiunta una riga [indice H3, indice poligono] per ogni cella trovata, in questo modo quando il client richiede le aree verdi di un determinato indice H3 noi sappiamo già quali sono andandoli a prendere nella tabella.



## 6 Client

### 6.1 Struttura ambiente di sviluppo e applicazioni React utilizzate



#### 6.1.1 Apis

Qua vengono contenuti i file per la gestione delle chiamate api, nel nostro caso al nostro server e a un'api che restituisce coordinate in base al nome.

Avendo utilizzato axios all'interno di questi file vi sono le configurazioni necessarie per quest'ultimo che vengono restituite a richiesta in una nuova istanza di axios in modo da non doverle definire in tutta la logica.

#### 6.1.2 Componenti

Qua sono definite cartelle di file dove si trovano tutti i componenti utilizzati dalla nostra applicazione come le parti della mappa, le liste, la barra di navigazione i bottoni eccetera. Molti di questi sono completamente dinamici, richiedendo ogni volta i dati da dover essere inseriti come vedremo dopo, altri statici come i bottoni che eseguono sempre la stessa azione.

Per lo stile dei componenti ho deciso di non utilizzare classi css ma una libreria molto utilizzata al momento che si chiama “styled-components” che permette di definire in costanti elementi html applicandogli uno stile da noi definito e poter utilizzare questi all'interno della nostra struttura.

```
const Container = styled.div`  
  width: 75%;
```

```
margin: auto;
`;
```

### 6.1.3 Hooks

Gli hooks sono delle funzioni simili a quelle javascript che permettono di “aggranciarsi” a stati e funzionalità riutilizzabili seguendo il ciclo di vita del componente a funzione in modo da non dover necessariamente creare un componente di classe.

Hanno due regole principali: non usarli all’interno di loop o condizioni, ma solamente a top level in modo da essere sicuri che vengano chiamati sempre nello stesso ordine a ogni render del componente e non chiamare gli hooks dentro funzioni javascript ma solo funzioni di react oppure in hooks customizzati.

Appunto React permette di creare hooks personalizzati come vedremo successivamente, ma ne fornisce di standard per funzionalità importanti nella vita dei componenti. Tra quelli che ho utilizzato ci sono:

**useState:** ritorna una variabile di stato e una funzione per modificarla.

```
const [state, setState] = useState(initialState);
```

**useEffect:** prende una funzione da lanciare in un deciso momento e un array di valori per cui lanciarla al loro cambiamento, invece se questo array è vuoto viene lanciata al completo rendering del componente.

Per evitare errori in caso di smontaggio e rimontaggio di componenti la funzione passata deve ritornare una seconda funzione chiamata “di pulizia”, per fare un semplice esempio se voglio avviare un timer alla creazione del mio componente che aumenta un intero ogni secondo, se il componente venisse smontato e rimontato verrebbe creato un nuovo timer, questo causerebbe l’aumento dell’intero di due volte ogni secondo. Per questo viene ritornata una funzione che cancella il timer, in modo da averne solamente uno in esecuzione.

```
useEffect( () => {
  setInterval(() => {
    console.log("loading");
    setLoadingStatus(loadingStatus + ".");
  }, 1000);
}, []);
```

```

    }, 1000);

    return function cleanup() {
        clearInterval(loop);
    };
}, [loadingStatus] );

```

**useContext:** prende un oggetto contesto di React (vedremo sotto cosa sono) e ne restituisce il valore corrente.

```
const value = useContext(MyContext);
```

**useRef:** restituisce un oggetto di referenza mutabile la cui proprietà `.current` è inizializzata dall'argomento (`initialValue`). L'oggetto restituito persisterà per l'intera durata del componente. L'ho utilizzato per esempio per accedere quando mi serviva a un componente figlio.

```

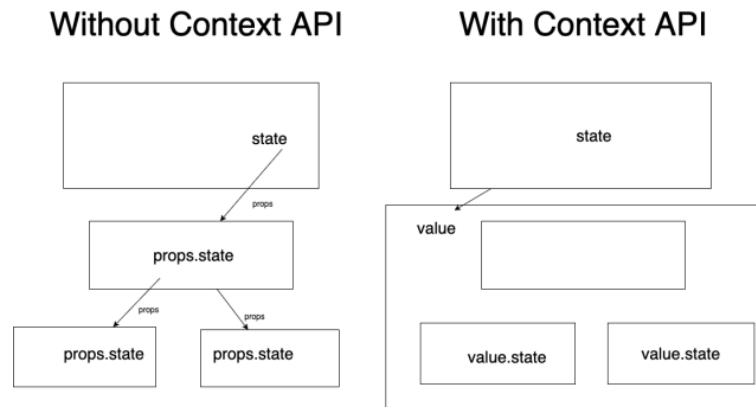
function TextInputWithFocusButton() {
    const inputEl = useRef(null);
    const onButtonClick = () => {
        // `current` points to the mounted text input element
        inputEl.current.focus();
    };
    return (
        <>
            <input ref={inputEl} type="text" />
            <button onClick={onButtonClick}>Focus the input</button>
        </>
    );
}

```

#### 6.1.4 Contesti

In React i dati vengono generalmente passati top-down dal padre al figlio, questo viene realizzato utilizzando i “props” visti precedentemente, questo potrebbe diventare però poco maneggevole in caso ci fossero troppi componenti che richiedono un

certo tipo di prop. I contesti forniscono quindi un modo per passare valori tra i componenti senza doverli passare esplicitamente per ogni livello dell'albero.



Possiamo quindi vederli come dei dati globali all'interno della struttura dei componenti e sono molto utili per esempio per gestire un utente autenticato, i temi dell'applicativo (dark, light), per la lingua scelta o come vedremo in seguito per la mia interattività con la mappa.

Il contesto necessita di due oggetti: un contesto vero e proprio e un provider, quest'ultimo è un componente wrapper che definisce la possibilità dei figli di accedere ai valori scelti del contesto, nell'esempio vediamo come è realizzato il mio provider per l'utente autenticato.

```
export const AuthProvider = ({ children }) => {  
  const [auth, setAuth] = useState(null);  
  
  return (  
    <AuthContext.Provider value={{ auth, setAuth }}>  
      {children}  
    </AuthContext.Provider>  
  );  
};
```

React offre quindi due metodi uno per la creazione del contesto

```
const AuthContext = createContext({});
```

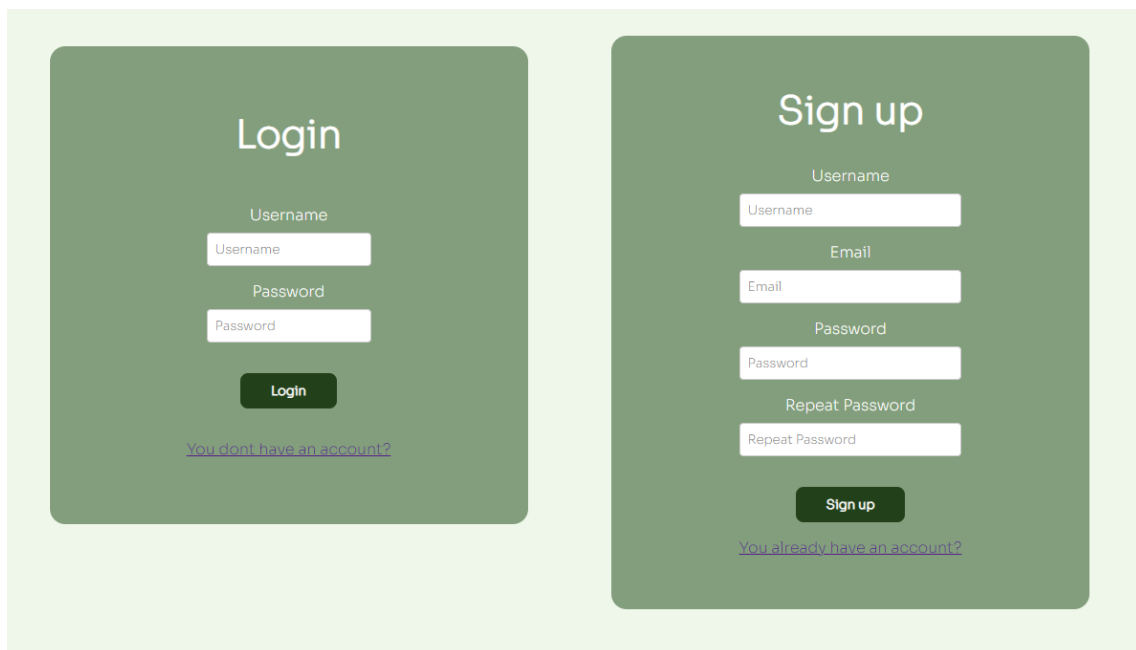
E un hook per il suo utilizzo

```
useContext(AuthContext);
```

Io per utilizzarlo in maniera più pulita e riutilizzabile ho creato un mio hook personalizzato che utilizza useContext passandogli AuthContext e ci ritorna il suo valore, in questo modo in devo importare ogni volta questi due ma solamente il mio hook.

```
const useAuth = () => {  
  return useContext(AuthContext);  
};
```

## 6.2 Autenticazione



The image displays two distinct forms for user authentication, presented side-by-side on a light green background. The left form, titled 'Login', features a dark green background with white text. It includes input fields for 'Username' and 'Password', a dark green 'Login' button, and a link for 'You dont have an account?'. The right form, titled 'Sign up', also has a dark green background with white text. It includes input fields for 'Username', 'Email', 'Password', and 'Repeat Password', a dark green 'Sign up' button, and a link for 'You already have an account?'. Both forms are visually clean and modern, using a consistent color scheme of dark green on a light green background.

Figura 1: form per l'autenticazione e registrazione.

Abbiamo appena visto che per il mantenimento dell'utente in tutta l'applicazione viene utilizzato un contesto, ovviamente il provider va a inglobare l'intera applicazione dando la possibilità di utilizzare i valori del contesto da tutti i componenti.

Utilizzando quindi l'hook useAuth i nostri componenti possono accedere all'utente autenticato di cui vengono salvati l'username, l'immagine, il ruolo e il token di accesso JWT di cui si è parlato precedentemente. Tutti i componenti possono quindi sapere di chi si tratta, caricare o permettere in base al ruolo e fare richieste al server con identità.

Oltre a questi ho implementato altri oggetti per la sicurezza e la persistenza dell'autenticazione. Come abbiamo visto in precedenza abbiamo salvato come cookie http il nostro token di refresh, ho quindi creato un hook useRefreshToken che mette a disposizione una funzione refresh che richiede al server un nuovo token di accesso e lo va a memorizzare nel contesto oltre che a ritornarlo.

Questo hook viene utilizzato in un altro mio hook useAxiosPrivate, che svolge lo stesso compito di useAxios visto in precedenza ma aggiungendo alla configurazione della chiamata api l'header "Authorization" con il nostro token di accesso. Come visto prima il token di accesso non viene più accettato dopo 30 secondi dalla sua creazione, come si può intuire in caso la chiamata non andasse a buon fine con errore 401 o 403 (non autorizzato) viene utilizzata la funzione refresh fornita da useRefreshToken e successivamente rifatta la richiesta con il nuovo token restituito. La chiamata viene rifatta mettendo come guardia dello useEffect il valore auth nel contesto che cambierà con il nuovo token.

Inoltre ho creato due componenti che vengono utilizzati per wrappare le nostre route, ovvero eseguono e/o fanno controlli prima di aprire la pagina richiesta. PersistentLogin contiene tutte le route e quello che fa è, in caso non ci fosse un valore nel contesto di autenticazione, verificare la presenza del nostro cookie e nel caso utilizzarlo per autenticarsi automaticamente. Questo come dice il nome ci permette di mantenere l'autenticazione anche chiudendo il nostro browser.

```
<Route element={<PersistentLogin />}>
  <Route path="/" element={<Home />} />
  <Route path="map" element={<MapPage />} />
  <Route path="about" element={<About />} />
  <Route path="login" element={<Login />} />
  ...
</Route>
```

RequireAuth vuole invece come prop un array di ruoli accettati per le route che contiene mandando la navigazione alla pagina "non autorizzato" se siamo loggati e il nostro ruolo non è contenuto nell'array oppure alla pagina di autenticazione nel caso non fossimo autenticati.

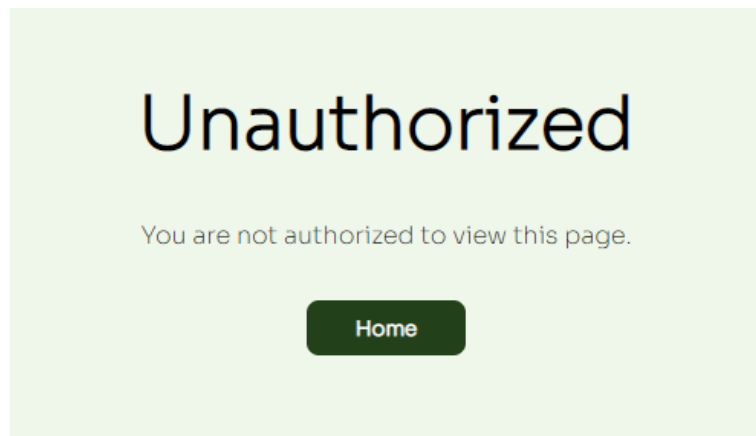


Figura 2: pagina non autorizzato.

```
<Route element={<RequireAuth allowedRoles={["admin"]} />}>  
  <Route path="admin" element={<Admin />} />  
</Route>
```

```
<Route element={<RequireAuth allowedRoles={["admin", "user"]} />}>  
  <Route path="/user" element={<User />} />  
  <Route path="/user/:username" element={<Profile />} />  
  <Route path="/feeds" element={<Feeds />} />  
</Route>
```

## 6.3 Pagina principale

La pagina di arrivo al sito si presenta con la navigazione e una preview gif per visualizzare le parti del sito che cambia in base al tipo di dispositivo.

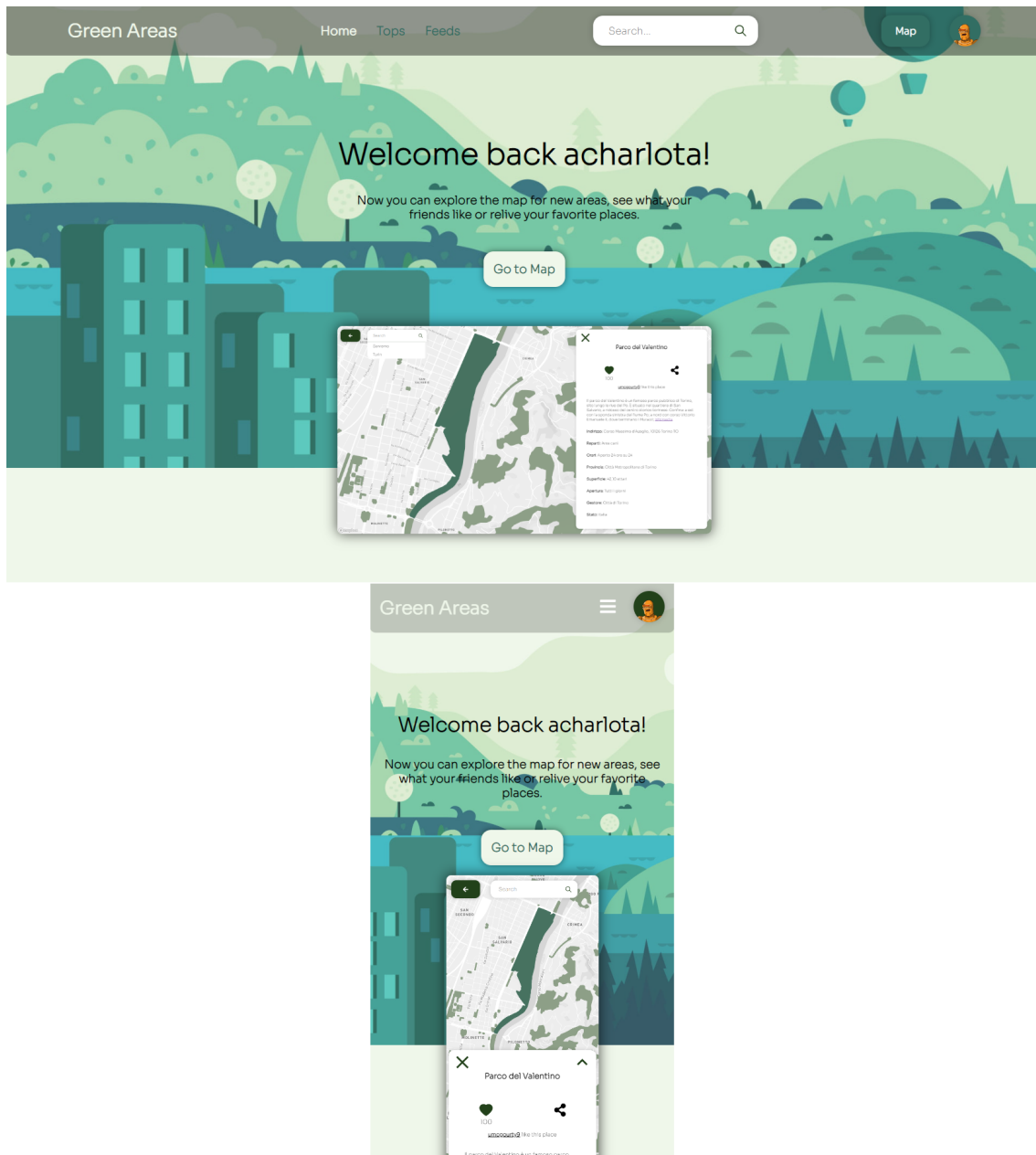


Figura 3: sopra la homepage per pc, sotto per mobile.



### 6.3.1 Ricerca globale

Nella navigazione è presente una ricerca globale che permette di cercare all'interno del server tutti gli utenti e i parchi e ovviamente accedere velocemente alla mappa o alla pagina dell'utente.

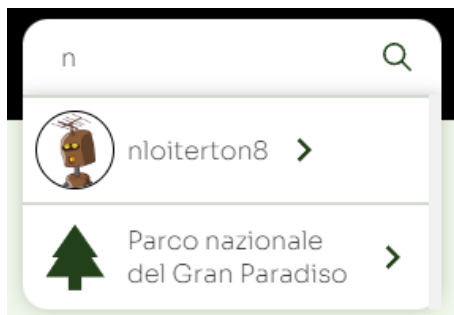


Figura 4: campo di ricerca nella navigazione.

## 6.4 La mappa

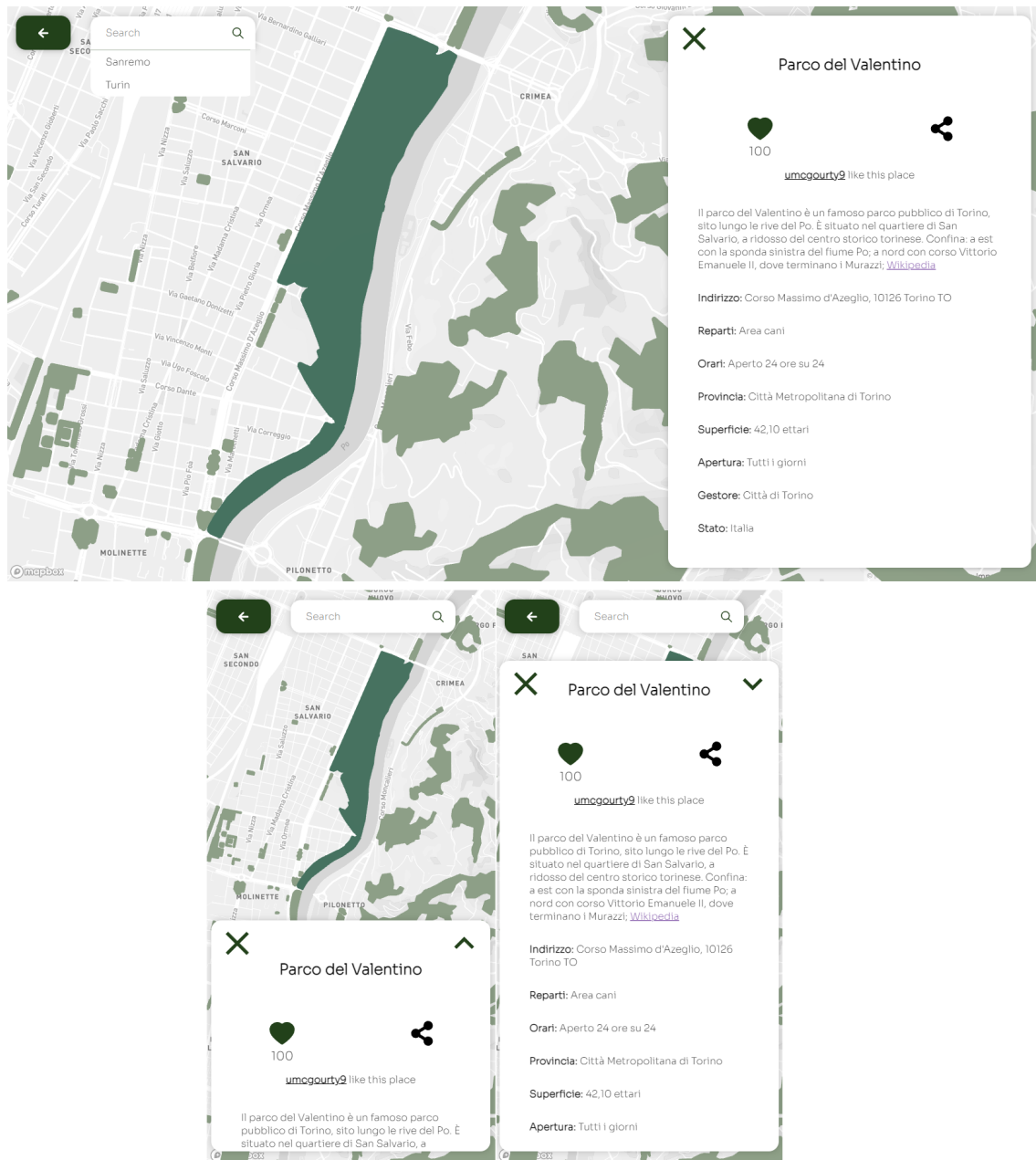


Figura 5: sopra la mappa su pc, sotto la mappa su mobile

La pagina della mappa è composta da tre componenti (mappa, pannello di dettaglio del poligono e ricerca) ed un contesto per la comunicazione tra questi.

Il nostro contesto funziona allo stesso modo di quello per l'autenticazione, ma utilizza l'hook useMap e mette a disposizione: uno stato per il poligono selezionato, ovvero il poligono su cui siamo concentrati sulla mappa e la sua funzione di cambiamento, uno stato per la posizione in cui vogliamo spostarci più la sua funzione

che collegano la mappa dal campo di ricerca e per finire una funzione di redirect, utilizzata per la condivisione tramite link tra utenti diversi.

#### **6.4.1 Campo di ricerca**

È un componente a funzione, alla sua creazione richiede attraverso useAxios al server le città che si trovano nel database. Accede al contesto tramite useMap, al suo focus ci mostra le città come risultati e al click di una città va a modificare il contesto di mappa mettendo le coordinate della città scelta. Digitando nel nostro campo di ricerca, invece, viene fatta una chiamata alla nostra api di geolocalizzazione che in base alla nostra ricerca restituisce posti con il medesimo nome, cliccando su un risultato inserisce le coordinate nel contesto.

#### **6.4.2 Pannello di dettaglio**

La presenza di questo componente all'interno della pagina è determinata dal valore del contesto “poligono selezionato” (settato dalla mappa), ovvero si vede solamente se questo valore non è nullo. Alla sua creazione richiede al server le informazioni dettagliate relative al poligono, se l'utente è autenticato utilizza useAxiosPrivate altrimenti useAxios, come visto nella parte relativa al server riceve in risposta i like totali del poligono e nel caso di chiamata autenticata anche le persone che seguiamo che hanno messo like alla stessa area urbana. Con il bottone di like è possibile mettere o togliere il mi piace e con il bottone di share viene copiato il link di redirect all'area verde selezionata.

#### **6.4.3 Mappa**

La mappa è il componente “principale” in questo progetto, è stato infatti il punto di partenza del progetto in quanto essenziale e necessita di “imparare” una libreria molto ampia e complessa come deck.gl.

Contiene diversi stati e metodi per la gestione delle aree verdi, prima di essere montata la pagina della mappa che contiene i tre componenti richiede al server gli indici H3 per cui dovremo richiedere al server le aree, questo riduce al minimo le chiamate al server durante l'esplorazione della mappa, questi indici vengono mappati

in una nuova struttura dati Map in cui la chiave è l'indice e il valore è un booleano che indica il caricamento effettuato o no, in questo caso mappato a false. La mappa mantiene questa struttura.

Altri stati sono un array di layer Deck.gl, una struttura dati Set che contiene gli id delle aree verdi caricate, questo è necessario in quanto più aree H3 possono toccare un'area verde. Ancora, uno stato che contiene un layer indipendente che identifica l'area selezionata, ci serve per evidenziare quest'ultima; infine uno stato per la viewState, un oggetto che contiene i parametri inerenti a dove si trova la view sulla mappa(lat, long, zoom ...) , ci permetterà di fare calcoli basati sulla nostra view.

Qua entra in gioco il nostro contesto: se il valore dell'area seleziona è nullo allora i parametri della viewState saranno la nostra posizione o in caso non ci fosse i valori di default, nel caso ci fosse un valore, che sarebbe un oggetto ovvero il nostro poligono, lo stato sarebbe ritornato da un metodo calculatePolyView.

Questo è uno dei primi metodi della mappa, prende in input delle coordinate, che sono sempre un array di due elementi ([latitudine, longitudine]) e il valore dell'area inteso come area geometrica del poligono; restituisce una view in cui lo zoom è calcolato tramite l'area: più il poligono è piccolo, più lo zoom sarà grande e latitudine e longitudine in questo modo: avendo il pannello di dettaglio su un lato, o sotto in caso di mobile, non vogliamo che il poligono sia centrale allo schermo ma leggermente spostato da non venire coperto, vado quindi a togliere qualcosa alla latitudine, o aggiungere alla longitudine su mobile, questo qualcosa è determinato da una formula che permette di calcolare la differenza di coordinate in base a una differenza di pixel, in questo modo possiamo andare a fare uno spostamento basato sulla grandezza dello schermo rendendo il tutto reattivo. La formula si basa su un'altra formula definita dalla documentazione di deck.gl, dice che un'unità comune è equivalente a  $2^{**}zoom$ .

Abbiamo due useEffect, uno eseguito alla creazione che in caso il valore nel contesto del poligono selezionato non fosse nullo, inserisce nello stato dell'area selezionata visto prima, il layer per il poligono; un'altro eseguito sulla guardia del valore del contesto della posizione, modificato dalla ricerca, questo cambia la viewState spostandosi sul luogo desiderato.

Abbiamo quindi visto che i tre componenti utilizzano il contesto per comunicare e definire cosa far vedere, in particolare la mappa lo utilizza per sapere da dove deve iniziare, questo ci servirà successivamente per l'interattività con le pagine social dove vi saranno bottoni per andare a vedere l'area verde sulla mappa, oppure nel caso di accesso alla mappa tramite il link di share dell'area.

Terminata la parte relativa al contesto la mappa esegue operazioni per conto suo per il caricamento dell'area, nel mio caso ho utilizzato la libreria H3 per eseguire un prefetching sulla mappa e quindi caricare oggetti fuori dalla mia area visiva (view) in modo che l'utente non debba subire caricamenti a schermo.

Il prefetching viene eseguito con più metodi per essere più leggibile e usabile ma qua lo vediamo come un unico blocco. Tramite lo zoom della nostra viewState viene calcolata la risoluzione da utilizzare e quindi la grandezza delle aree (la cosa migliore è che l'area copra quasi lo schermo), con latitudine, longitudine correnti e la risoluzione passati al metodo della libreria H3 `geoToH3` viene ricavato l'indice dell'area h3 su cui ci troviamo. Calcoliamo adesso gli anelli usando il metodo della stessa libreria `kRing` passando l'indice ricavato e due come numero di anelli.

In seguito per ogni indice se all'interno della struttura Map per quella key esiste e il valore di caricamento è falso viene messo a vero e vengono richieste al server le aree contenute. Queste nuove aree vengono filtrate in base a quelle già caricate che si trovano nella Set, successivamente questa viene aggiornata e viene creato un nuovo layer con i poligoni appena caricati che viene aggiunto allo stato dei layer presenti sulla mappa e visibili.

Al click su una delle aree caricate viene aggiornato il valore nel contesto per l'area selezionata in modo che il pannello di dettaglio si monti con le giuste impostazioni, viene generato il layer indipendente per il poligono selezionato in modo da evidenziare l'area sulla mappa e con il metodo visto prima `calculatePolyView` viene modificata la viewState per metterla in primo piano.

#### **6.4.4 Redirect**

Abbiamo visto che le aree possono essere condivisi tramite link, essi hanno questa forma: `.../redirect?id=<id dell'area>` Su questa route si trova un componente

che ha lo scopo di richiedere al server il poligono dell'area con determinato id e di inserirlo nel contesto della mappa come poligono selezionato, in seguito esegue un redirect alla pagina della mappa che come visto prima si accorgerà della presenza del poligono selezionato e si posizionerà a dovere, anche il pannello di dettaglio se ne accorgerà aprendosi e chiedendo al server i dettagli del poligono.

## 6.5 Social

La funzione social di questo sito permette di mantenere i propri mi piace, di seguire altri utenti e di vedere i mi piace dei medesimi. Dato che le pagine si accomunano molto i componenti sono stati creati in modo dinamico ed è stato implementato un caricamento a pagine “lazy loading” in modo da non fare chiamate troppo pesanti al server e/o caricare dati inutili.

Vi sono tre componenti principali: l'header del profilo, le tabs ovvero i pulsanti per cambiare vista e le liste, oltre a piccoli componenti come titoli e pulsanti. L'idea è quella di inserire gli oggetti dando informazioni e lasciare che siano loro a gestire il tutto.

### 6.5.1 Tabs

Tabs è un componente wrapper che prende al suo interno componenti che andranno scambiati per la visualizzazione, quest'ultimi devono avere come prop “tabname” che sarà il nome del bottone associato alla vista del componente. Quindi per ogni elemento genera un bottone che quando viene cliccato cambia uno stato ad interi, lo stato indica quale oggetto deve essere caricato nella vista.

### 6.5.2 List

L'idea è quella di creare una lista generale che prende come input l'api su cui fare la chiamata al server per i dati e un elemento che preso un singolo dato come prop ne definisce la struttura, questo sarà l'elemento utilizzato per ogni singola riga.

```
<List api={` /social/likes/${username}`} element={LikeItem} />
```

### 6.5.3 Lazy Load e Infinite Scroll

La list non fa una semplice chiamata per i dati al server ma come visto nella parte backend fa più chiamate a pagine richiedendo un numero limitato di dati alla volta. Per fare ciò ho creato un hook useLazyLoader che utilizzando un intero “pagina” e la query che abbiamo passato alla lista prima carica la porzione degli elementi ogni volta che le due variabili cambiano utilizzando useEffect con guardia. Per ricreare l'effetto dello scroll infinito il caricamento successivo è definito appunto dal cambiamento e più precisamente dall'aumento del numero della pagina eseguito grazie a una referenza sull'ultimo elemento della lista per osservare quando entra nella nostra view. Appena l'elemento viene visualizzato a schermo quindi viene aumentato il numero della pagina richiedendo all'hook di caricare nuovi dati e inserirli nella lista che aggiorna la referenza del nuovo ultimo elemento.



## 6.5.4 Pagine

### Utente autenticato e profilo di altri utenti

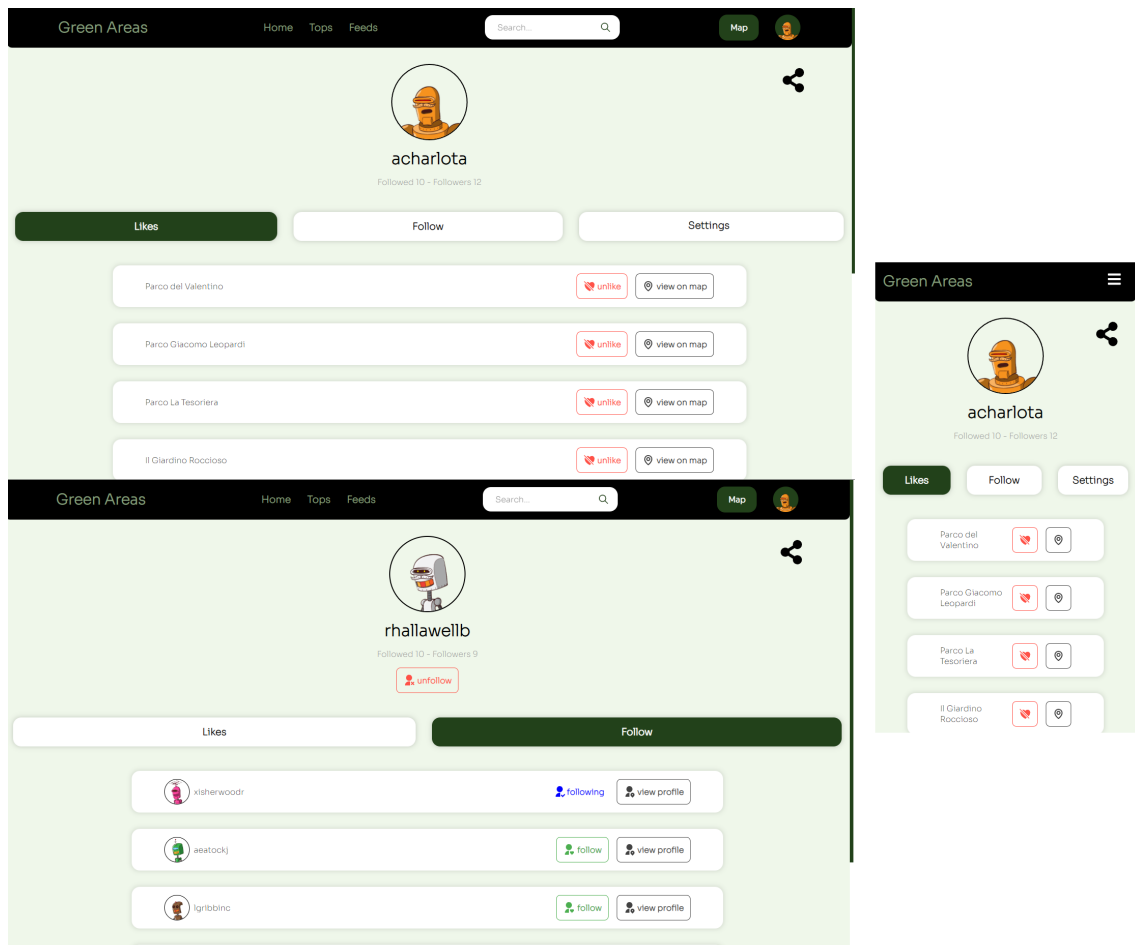


Figura 6: sopra la pagina personale, sotto la pagina di altri utenti, a destra la versione mobile

Utilizzano i medesimi componenti citati prima ma nella propria pagina è possibile togliere like e smettere di seguire. Il bottone per vedere il poligono sulla mappa utilizza la funzione redirect contenuta nel contesto della mappa.



## Feeds

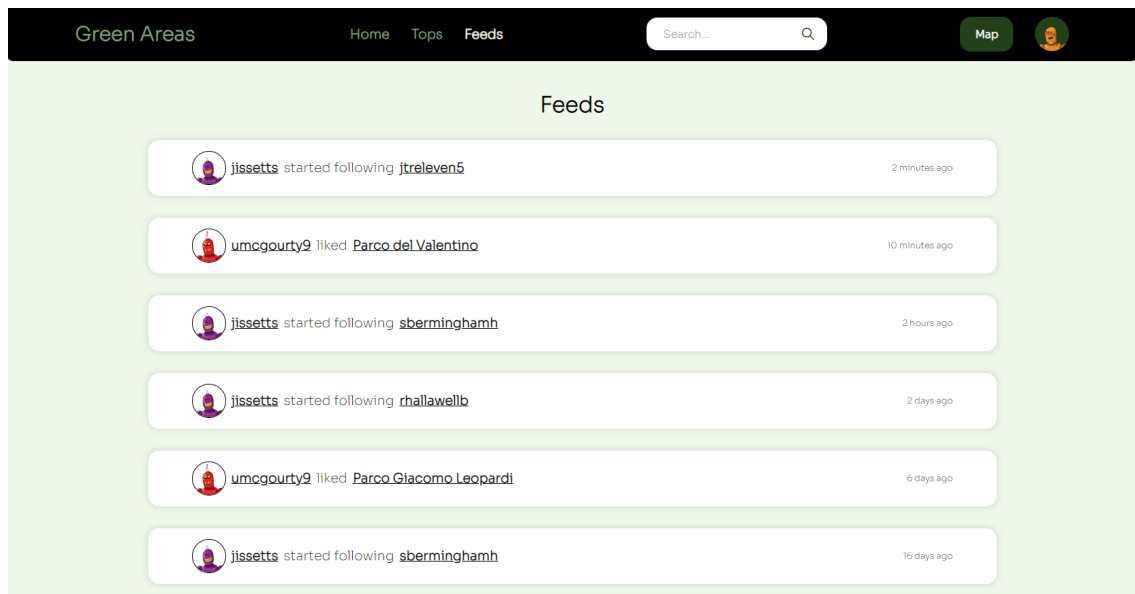


Figura 7: pagina delle attività recenti

Una pagina che contiene il componente lista visto prima e permette di vedere le ultime azioni delle persone che seguiamo.

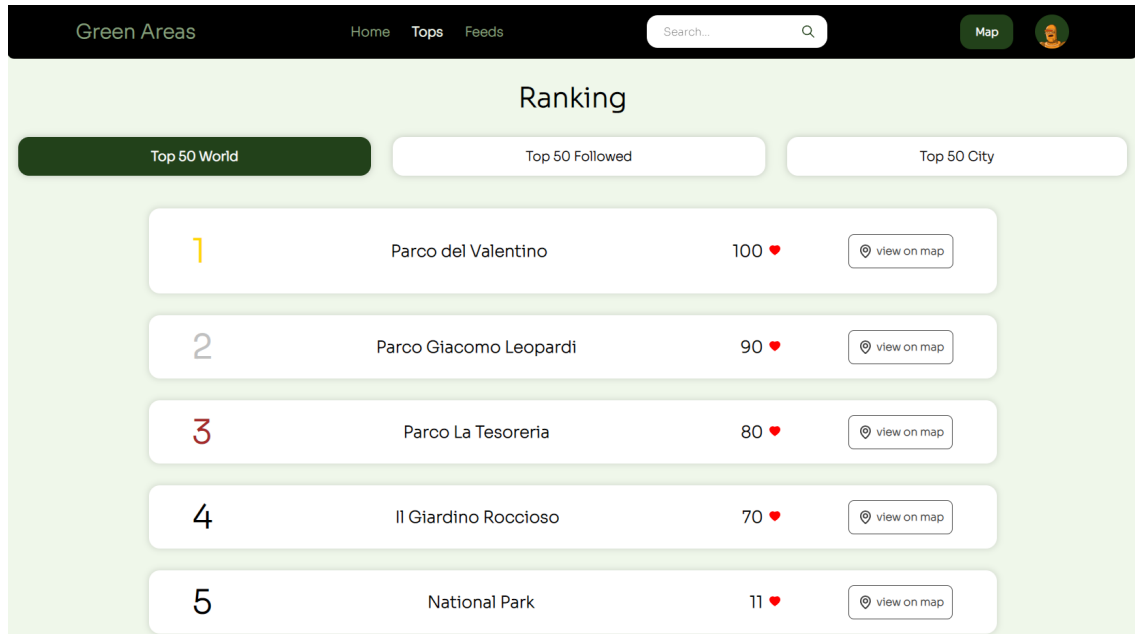


Figura 8: la pagina delle classifiche

Una pagina che utilizza tabs e list per far visualizzare tre classifiche delle aree verdi più votate globalmente, tra i seguiti e per città.

## 7 Conclusioni

Il progetto di questo stage era incentrato sulla "costruzione" di una web application in maniera full stack, ovvero coprendo sia la parte server backend che la parte client frontend.

Durante questo stage sono riuscito a vedere molte cose anche nello specifico e soprattutto nel loro contesto di utilizzo e quali scelte comportano ma, ho anche capito che ho molte lacune e possibilità di migliorare riguardo ad aspetti più specifici e complessi di un framework come React che, per essere utilizzato al meglio, necessita di molta più conoscenza delle sue funzionalità. Sicuramente in futuro continuerò ad utilizzare questo strumento in tutte le sue sfumature (Next.js) per riuscire a padroneggiarlo al meglio.

Ringrazio inoltre il Professor Rossano Schifanella per i consigli implementativi e di scelta delle tecnologie.

### 7.1 Obbiettivi raggiunti

- Realizzare una web application con il framework React e comprenderne le funzionalità di base(componenti, stati, navigazione, ecc...) e avanzate (contesti, hook personalizzati, memorizzazione, ecc...).
- Costruire componenti dinamici e riutilizzabili in tutta l'applicazione per garantire una facile modifica.
- Realizzare un server che fornisce api con il micro-framework Flask in Python gestendo tutte le sue caratteristiche di modularità (Blueprints), controlli, gestione di CORS, route private (autenticazione) e gestione di un database SQL.
- Creare un database spaziale con PostGIS e comprenderne i metodi per la gestione e l'acquisizione di forme geometriche nello spazio lato server con l'utilizzo del formato GeoJSON.
- Comprendere il funzionamento dei JSON Web Tokens e utilizzarli per la comunicazione autorizzata tra client e server e per gestire il mantenimento del-

l'autenticazione lato client agli accessi futuri.

- Comprendere il corretto utilizzo di Axios lato client per richiedere dati al server (configurazione, cancellazione di richieste multiple, ecc...)
- Utilizzare la libreria Deck.gl per la visualizzazione su browser di grandi quantità di dati con annesse necessità:
  - Comprendere il funzionamento della libreria (componente React, layers, view, render, ecc...)
  - Caricare i dati in modo incrementale eseguendo un pre-caricamento su quello che vorrà vedere l'utente
  - Ottimizzare l'utilizzo dei layer per non comportare render e calcoli inutili
  - Fornire all'utente un'interazione sulla mappa immersiva ma anche reattiva
- Realizzare pagine social-interattive attorno alla nostra mappa per la condivisione di posti piaciuti ad altri utenti e visualizzarli sulla mappa con un semplice click, seguire altri utenti per rimanere aggiornati sulle loro attività sulla piattaforma.
- Utilizzare alcune pratiche di Interazione uomo e macchina per creare un'applicazione usabile:
  - Caricamenti veloci e placeholder in caso di connessione lenta, anche utilizzando LazyLoading su scroll di pagina molto lunghi.
  - Stile omogeneo per tutto il sito con pochi colori che indicano lo scopo del componente.
  - Una facile navigazione (facile struttura del sito) anche con una ricerca globale.
  - Pochi click per eseguire le proprie azioni, facile utilizzo della mappa per trovare ciò che cerchiamo.
  - Design responsive che si adatta allo schermo.

## 7.2 Sviluppi futuri

- In Deck.gl esiste un layer che esegue un caricamento di dati poco alla volta, a porzioni, proprio come io ho realizzato ma dava la possibilità di separare le aree solamente tramite coordinate OMS (x,y,z) e non altri indici. Nella nuova versione della libreria viene aggiunta la possibilità di implementare questo layer e personalizzarlo creando il proprio sistema di indicizzazione, questo darebbe la possibilità di utilizzare l'indicizzazione H3 in maniera già ottimizzata per Deck.gl e realizzato dagli sviluppatori della libreria senza dover creare da sé il proprio sistema di pre-caricamento.
- Avevo l'idea inserire un sistema di rating con recensioni da visualizzare nel pannello di dettaglio del poligono e ovviamente inserire un controllo intelligente lato server per la ricerca di commenti volgari o non appropriati.
- Aggiungere la navigazione in tempo reale e quindi di poter calcolare percorsi ottimizzati in base alla posizione, infatti utilizzando Mapbox c'è la possibilità di calcolare i percorsi e grazie a Deck.gl di visualizzarli sulla mappa.
- Rendere la vista della mappa più appagante magari con grafica personalizzata e modelli 3D.
- Aggiungere la possibilità di creare account partendo da altri come Google.