



1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

DISPEA
DIPARTIMENTO DI
SCIENZE PURE E
APPLICATE

Relazione Progetto d'Esame

Tecnologie WEB per la Gestione del Territorio

Sessione Autunnale 2024/2025

FindSPOT

AUTORI

Mattia Gasperoni

Matricola: 329235

Andrea Rossi

Matricola: 327661

Indice

1 Introduzione	3
1.1 Descrizione del Servizio	3
1.2 Obiettivi del Servizio	3
2 Architettura e Scelte Implementative	4
2.1 Componenti Software	4
2.1.1 Front-End	4
2.1.2 Back-End	4
2.2 Comunicazione tra componenti	4
2.3 Tecnologie adottate	4
3 Dati e Servizi Esterni	5
3.1 Fonte dei Dati Utilizzati	5
3.2 Struttura del file GeoJSON	6
3.3 Servizi Esterni Utilizzati	7
4 Documentazione dell'API RESTful	8
4.1 Elenco degli Endpoint	8
4.2 Dettagli degli Endpoint	8
4.2.1 GET /api/parcheggi	8
4.2.2 POST /api/parcheggi	9
4.2.3 PUT /api/parcheggi/:id	9
4.2.4 DELETE /api/parcheggi/:id	10
4.3 Testing degli Endpoint tramite Postman	11
4.4 Codici di Stato HTTP di Buona Riuscita	13
4.5 Codici di Stato HTTP per la Gestione degli Errori	13
5 Funzionamento del Web Service	14
5.1 Home Page	14
5.2 Pagina per Visualizzare la Mappa	15
5.3 Pagina per Aggiungere un Parcheggio	17
5.4 Pagina per Rimuovere un Parcheggio	18
5.5 Pagina per Modificare un Parcheggio	19

1 Introduzione

1.1 Descrizione del Servizio

FindSPOT è un'applicazione web per la visualizzazione e la gestione interattiva dei parcheggi nella regione Marche. L'app offre una mappa dinamica, su cui sono rappresentati visivamente i parcheggi, permettendo all'utente di interagire direttamente con essi.

Tramite l'interfaccia grafica, è possibile visualizzare i dettagli dei parcheggi, applicare dei filtri di ricerca, aggiungere nuovi parcheggi, modificare parcheggi esistenti oppure eliminarli dalla mappa.

1.2 Obiettivi del Servizio

Gli obiettivi principali del servizio **FindSPOT** sono i seguenti:

- Fornire una piattaforma accessibile via web per la consultazione interattiva dei parcheggi nella regione Marche.
- Permettere l'applicazione di filtri per facilitare la ricerca di parcheggi con caratteristiche specifiche (es. pubblici, gratuiti, asfaltati).
- Offrire strumenti semplici per l'aggiunta, la modifica e l'eliminazione dei parcheggi dalla mappa, in tempo reale.
- Promuovere il riutilizzo di Open Data e dimostrare come possano essere valorizzati attraverso tecnologie web moderne.
- Favorire la scalabilità del servizio in modo da poter essere adattato facilmente ad altri territori o dataset simili.

2 Architettura e Scelte Implementative

2.1 Componenti Software

2.1.1 Front-End

Il front-end dell'applicazione è interamente sviluppato in HTML, CSS e JavaScript puro, con il supporto della libreria **Leaflet.js** per la visualizzazione e l'interazione con la mappa. L'interfaccia utente è suddivisa in diverse pagine specifiche:

- `index.html` – home page e punto di accesso al servizio
- `map.html` – interfaccia per la visualizzazione dei parcheggi su mappa interattiva
- `add.html` – form per l'aggiunta di nuovi parcheggi;
- `remove.html` – interfaccia per la rimozione dei parcheggi esistenti
- `edit.html` – interfaccia per la modifica delle proprietà di un parcheggio

Ogni pagina è associata a un proprio file CSS e JavaScript dedicato, per garantire una migliore organizzazione del codice e uno stile coerente e personalizzato tra le varie sezioni dell'applicazione. In particolare, gli script JS gestiscono la logica dell'interazione con la mappa e la comunicazione con il back-end, mentre i CSS controllano l'aspetto visivo e la responsività dell'interfaccia.

2.1.2 Back-End

Il back-end è sviluppato con **Node.js** e **Express.js**. Funziona da Web service RESTful che fornisce i dati al front-end in formato JSON. Il server espone una serie di endpoint per il recupero, l'aggiunta, la modifica e la cancellazione dei parcheggi, che sono memorizzati in un file GeoJSON.

Le principali responsabilità del back-end sono:

- Caricare e manipolare il file `parcheggi.geojson`;
- Gestire richieste HTTP sui percorsi `/api/parcheggi`;
- Validare i dati ricevuti dal client;
- Generare ID univoci in base alle coordinate;

2.2 Comunicazione tra componenti

La comunicazione tra front-end e back-end avviene tramite chiamate HTTP asincrone (AJAX). Il front-end invia richieste GET, POST, PUT e DELETE agli endpoint RESTful esposti dal server Express.

Il formato dei dati scambiati è **GeoJSON**, ampiamente supportato per la rappresentazione di feature geospatiali. Le chiamate sono implementate tramite `fetch()` in JavaScript, e le risposte vengono poi elaborate per aggiornare dinamicamente la mappa e l'interfaccia utente.

2.3 Tecnologie adottate

Le principali tecnologie utilizzate nel progetto sono:

- **Node.js** – piattaforma JavaScript per l'esecuzione lato server;
- **Express.js** – framework per la gestione delle API RESTful;
- **Leaflet.js** – libreria open-source per mappe interattive;
- **HTML/CSS/JavaScript** – per la costruzione dell'interfaccia utente;
- **GeoJSON** – formato standard per la rappresentazione di dati geografici;
- **Open Data** – dataset pubblico contenente i parcheggi nella regione Marche.

3 Dati e Servizi Esterni

3.1 Fonte dei Dati Utilizzati

I dati utilizzati nel progetto provengono da **OpenStreetMap (OSM)**, una piattaforma open source che fornisce dati geografici liberi e modificabili da chiunque.

I dati relativi ai parcheggi sono stati estratti da OSM in formato **GeoJSON** e successivamente filtrati per includere soltanto i parcheggi situati nella regione **Marche**.

- **Fonte:** OpenStreetMap
- **Formato:** GeoJSON
- **File locale:** data/parcheggi.geojson

Link alla fonte dati:

- **OpenStreetMap:** <https://www.openstreetmap.org>
- **Sottoinsieme Open Data:** <https://overpass-turbo.eu>

3.2 Struttura del file GeoJSON

Il file `parcheggi.geojson` contiene la rappresentazione dei parcheggi sotto forma di oggetti **GeoJSON**, uno standard per la codifica di dati geografici strutturati in formato JSON.

La struttura base del file è quella di un oggetto `FeatureCollection`, che contiene un array di elementi `Feature`. Ogni feature rappresenta un singolo parcheggio.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [12.345678, 43.123456]
      },
      "properties": {
        "name": "Parcheggio Piazza XX",
        "access": "public",
        "fee": "no",
        "surface": "asphalt"
      }
    }
  ]
}
```

Spiegazione dei campi principali:

- **type**: indica il tipo di elemento GeoJSON (nel file principale è `FeatureCollection`).
- **features**: è una lista di oggetti `Feature`, ciascuno dei quali rappresenta un parcheggio.
- **geometry**: contiene la geometria del parcheggio, sia il `"type"` che può essere un Point o un Polygon che le `"coordinates"` definite come [longitudine, latitudine]).
- **properties**: contiene informazioni aggiuntive, personalizzabili, relative al parcheggio.

Proprietà utilizzate:

- **name**: Nome del parcheggio
- **access**: Tipo di accesso (`public` o `private`)
- **fee**: Indicazione se il parcheggio è a pagamento (`yes` o `no`)
- **surface**: Tipo di superficie (`asphalt`, `grass`, `gravel`)

Il file **GeoJSON** viene caricato dal server all'avvio dell'applicazione e può essere modificato dinamicamente attraverso le API RESTful per aggiungere, aggiornare o rimuovere parcheggi.

3.3 Servizi Esterni Utilizzati

Per la visualizzazione dei dati geografici e l'interazione con la mappa, viene utilizzata la libreria open source:

- Leaflet.js: <https://leafletjs.com>

Leaflet è una delle librerie più leggere ed efficienti per il rendering di mappe interattive sul web e permette la gestione di marker, popup, stili dinamici, livelli, e altro.

Non sono utilizzati altri servizi esterni in quanto i dati sono completamente gestiti localmente nel file GeoJSON.

4 Documentazione dell'API RESTful

Il back-end del progetto espone un Web service RESTful per la gestione dei dati dei parcheggi. Tutti gli endpoint accettano e restituiscono dati in formato **GeoJSON**, e supportano le principali operazioni CRUD: Create (POST), Read (GET), Update (PUT), Delete (DELETE).

4.1 Elenco degli Endpoint

Metodo	Endpoint	Descrizione	Dati richiesti
GET	/api/parcheggi	Restituisce tutti i parcheggi (filtrabili)	Query string opzionale
POST	/api/parcheggi	Aggiunge un nuovo parcheggio	Feature GeoJSON nel body
PUT	/api/parcheggi/:id	Modifica un parcheggio esistente	ID nell'URL + proprietà JSON
DELETE	/api/parcheggi/:id	Elimina un parcheggio	ID nell'URL

4.2 Dettagli degli Endpoint

4.2.1 GET /api/parcheggi

Restituisce tutti i parcheggi disponibili. È possibile filtrare i risultati usando parametri nella query string.

Esempio di richiesta:

- GET /api/parcheggi?access=public&fee=no

Filtri supportati: I filtri sono validati in base alle proprietà presenti nei dati. Il confronto è case-insensitive e con trim automatico degli spazi. Filtri accettati:

- **name:** nome del parcheggio
- **access:** tipo di accesso (es. `public`, `private`)
- **fee:** presenza di tariffe (es. `yes`, `no`)
- **surface:** tipo di superficie (es. `asphalt`, `grass`, `gravel`)

Risposta:

- **Successo (200):** GeoJSON con array di feature filtrate
- **Nessun risultato (200):** JSON con messaggio e array vuoto
- **Filtri non validi (400):** Errore con elenco filtri non riconosciuti

4.2.2 POST /api/parcheggi

Aggiunge un nuovo parcheggio. L'ID viene generato automaticamente nel formato `longitude_latitude` (arrotondato a 6 cifre decimali) e viene assegnato alla proprietà `@id`.

Validazioni eseguite:

- Presenza di geometria e coordinate
- Coordinate numeriche valide
- Range coordinate: longitudine [-180, 180], latitudine [-90, 90]
- Unicità della posizione (controllo duplicati)

Corpo JSON richiesto:

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [12.345678, 43.123456]
  },
  "properties": {
    "name": "Parcheggio Piazza XX",
    "access": "public",
    "fee": "no",
    "surface": "asphalt"
  }
}
```

Risposte possibili:

- **Successo (201):** Conferma creazione con ID assegnato
- **Dati invalidi (400):** Errore di validazione
- **Posizione occupata (409):** Parcheggio già esistente nelle stesse coordinate

4.2.3 PUT /api/parcheggi/:id

Aggiorna le proprietà di un parcheggio esistente. Solo le proprietà consentite vengono elaborate: `name`, `access`, `fee`, `surface`.

Corpo JSON (proprietà da aggiornare):

```
{
  "name": "Parcheggio aggiornato",
  "access": "private",
  "fee": "yes",
  "surface": "concrete"
}
```

Caratteristiche:

- Aggiornamento parziale: è possibile inviare solo le proprietà da modificare
- Validazione delle proprietà consentite
- Preservazione di proprietà non modificabili (`@id`, `coordinate`)

Risposte possibili:

- **Successo (200):** Conferma aggiornamento con proprietà modificate
- **Parcheggio inesistente (404):** ID non trovato
- **Dati invalidi (400):** Nessuna proprietà valida fornita

4.2.4 DELETE /api/parcheggi/:id

Elimina il parcheggio identificato dall'ID specificato. L'ID deve corrispondere esattamente al valore della proprietà `@id` del parcheggio.

Esempio:

- DELETE /api/parcheggi/12.345678_43.123456

Risposte possibili:

- **Successo (200):** Conferma eliminazione con ID rimosso
- **Parcheggio inesistente (404):** ID non trovato

4.3 Testing degli Endpoint tramite Postman

Per verificare il corretto funzionamento degli endpoint del web service, è stato utilizzato lo strumento Postman, un'applicazione che consente di inviare richieste HTTP a un server REST e di analizzarne le risposte. Di seguito vengono descritti i test eseguiti per ciascuna delle principali operazioni disponibili.

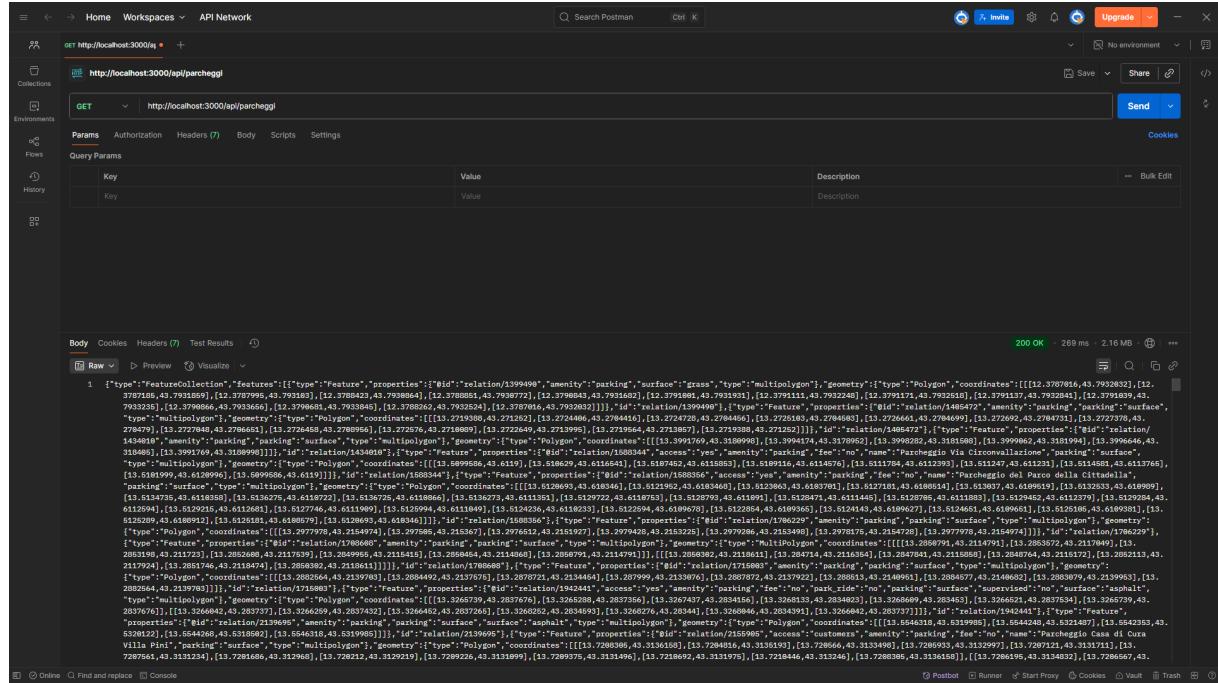


Figure 1: Richiesta GET per ottenere tutti i parcheggi

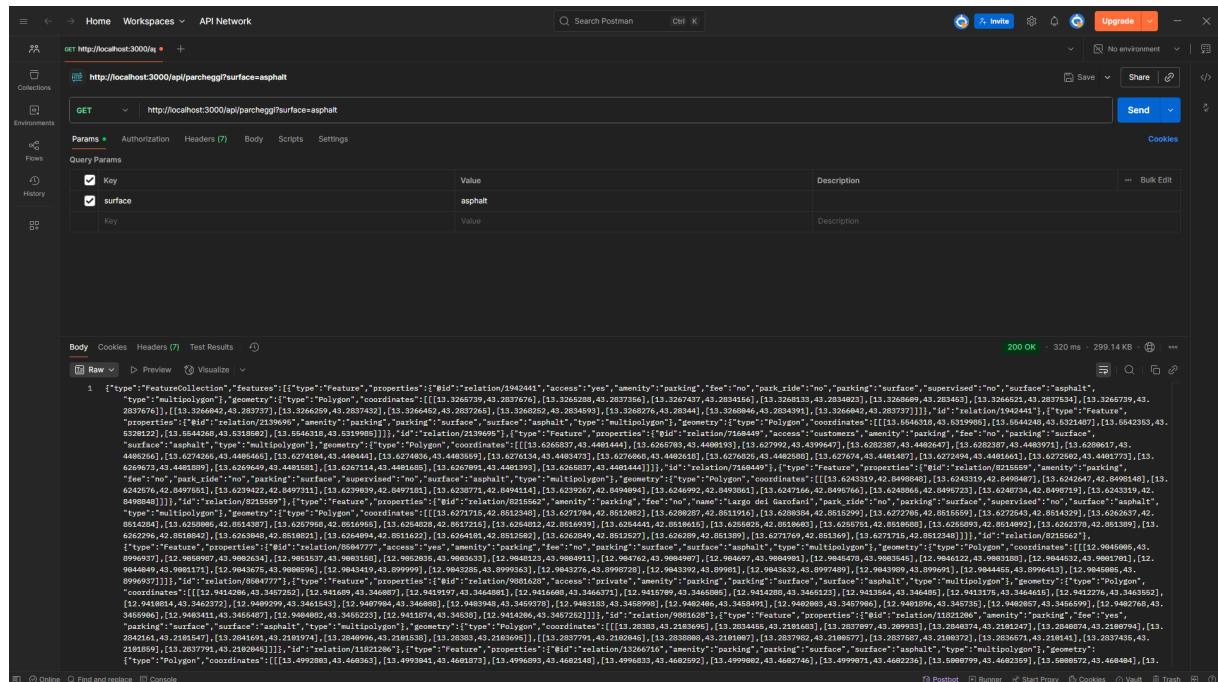


Figure 2: Richiesta GET con filtro per Superficie = Asfalto

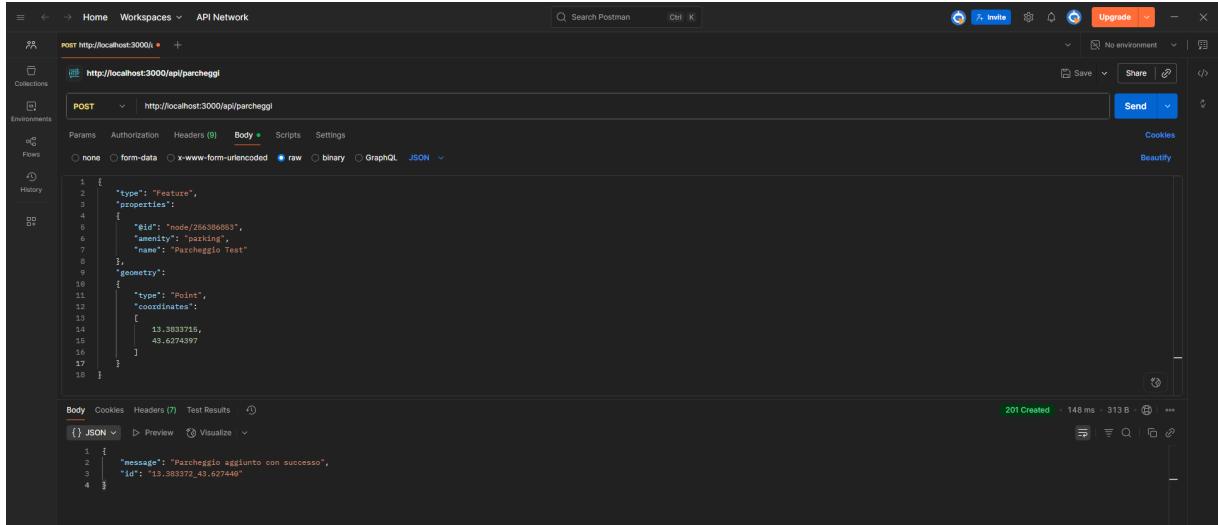


Figure 3: Richiesta POST per aggiungere un nuovo parcheggio

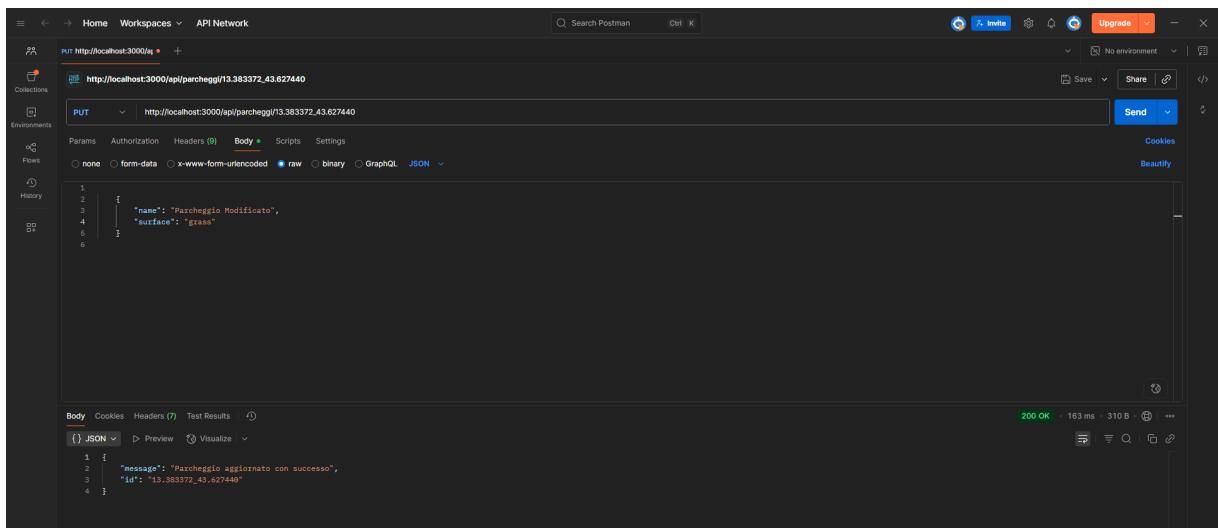


Figure 4: Richiesta PUT per aggiornare i dati di un parcheggio

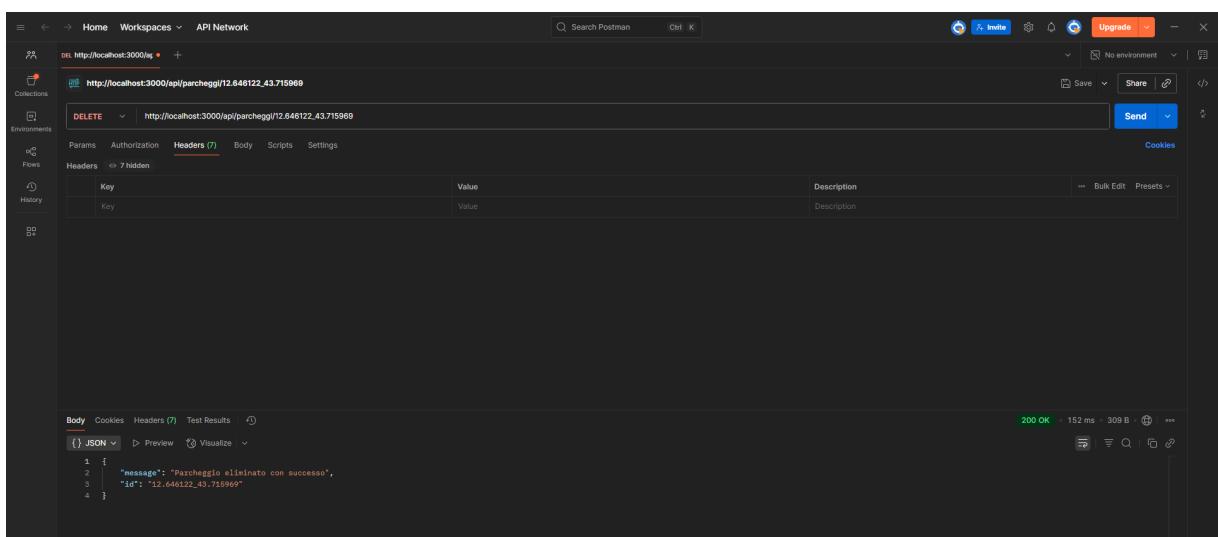


Figure 5: Richiesta DELETE per rimuovere un parcheggio dal sistema

4.4 Codici di Stato HTTP di Buona Riuscita

- 200 OK - Richiesta eseguita con successo.

Utilizzato per:

- GET /api/parcheggi – elenco dei parcheggi (anche quando il risultato è vuoto);
- PUT /api/parcheggi/:id – modifica di un parcheggio esistente;
- DELETE /api/parcheggi/:id – eliminazione di un parcheggio.

- 201 Created - Nuova risorsa creata correttamente.

Utilizzato per:

- POST /api/parcheggi – aggiunta di un nuovo parcheggio.

4.5 Codici di Stato HTTP per la Gestione degli Errori

Il server implementa una gestione centralizzata degli errori tramite un middleware Express che intercetta tutte le eccezioni non gestite esplicitamente nelle rotte. Inoltre, le rotte API gestiscono gli errori comuni e restituiscono messaggi JSON con un codice HTTP appropriato.

- 400 Bad Request - Dati non validi o incompleti nella richiesta.

Esempi:

- Corpo della richiesta mancante o non valido;
- Geometria o coordinate mancanti nella creazione di un parcheggio;
- Coordinate non valide;
- Parametri filtri non riconosciuti nella GET;
- Nessuna proprietà valida fornita per l'aggiornamento nella PUT.

- 404 Not Found - Risorsa non trovata.

Esempi:

- Il parcheggio richiesto non esiste;
- L'endpoint richiesto non è definito.

- 409 Conflict - Conflitto con lo stato corrente della risorsa.

Esempi:

- Tentativo di creare un parcheggio in una posizione già occupata.

- 500 Internal Server Error - Errore generico lato server.

Esempi:

- Errore nella lettura o scrittura del file `parcheggi.geojson`;
- Formato GeoJSON non valido nel file dati;
- Errore imprevisto durante l'elaborazione della richiesta.

5 Funzionamento del Web Service

5.1 Home Page

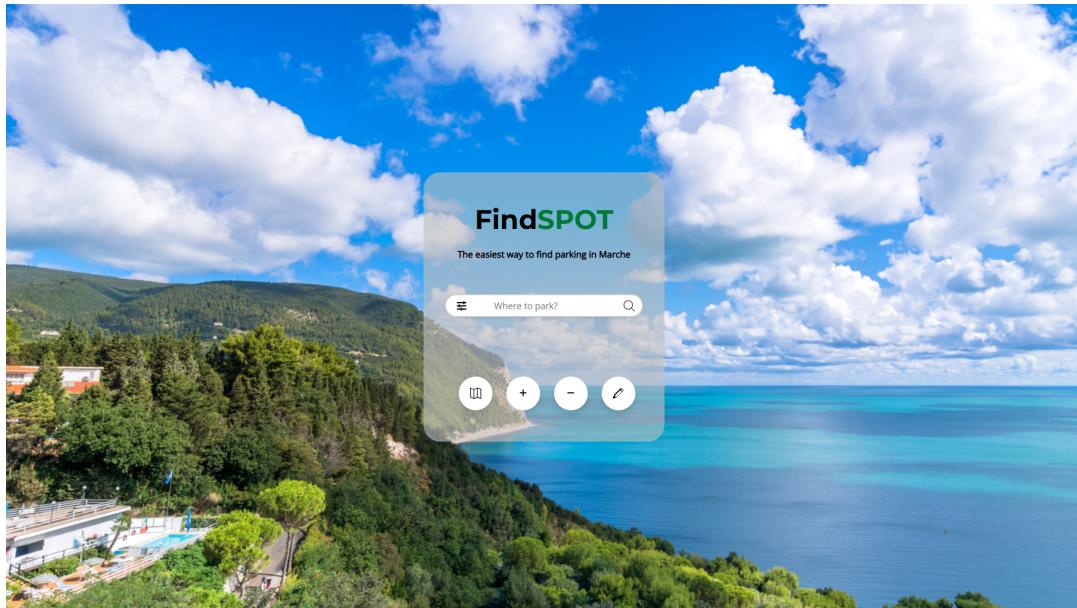


Figure 6: Home page del Web Service

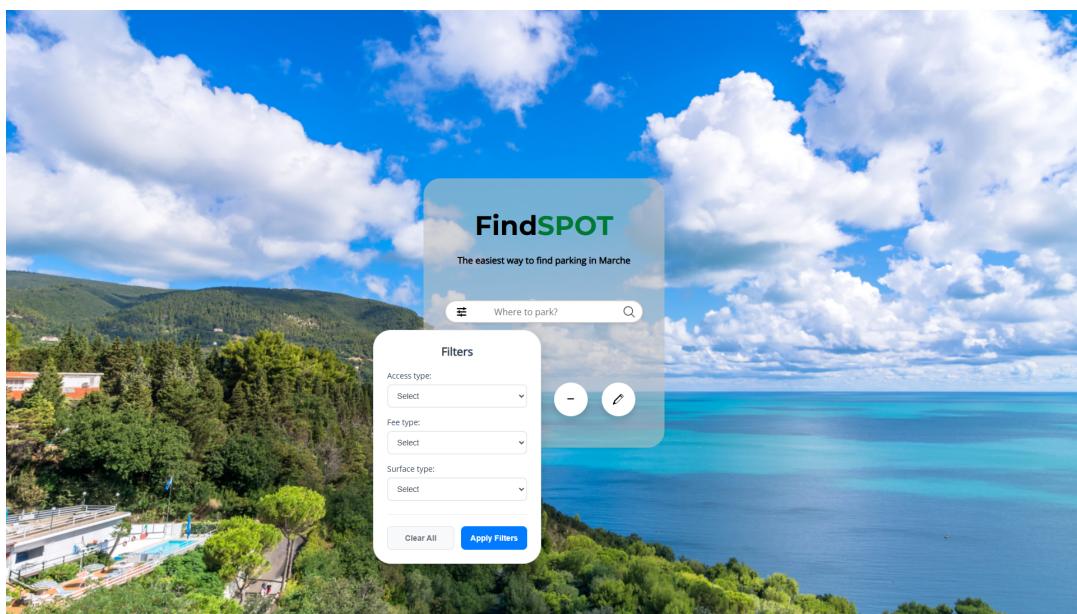


Figure 7: Pannello dei Filtri della Home Page

5.2 Pagina per Visualizzare la Mappa

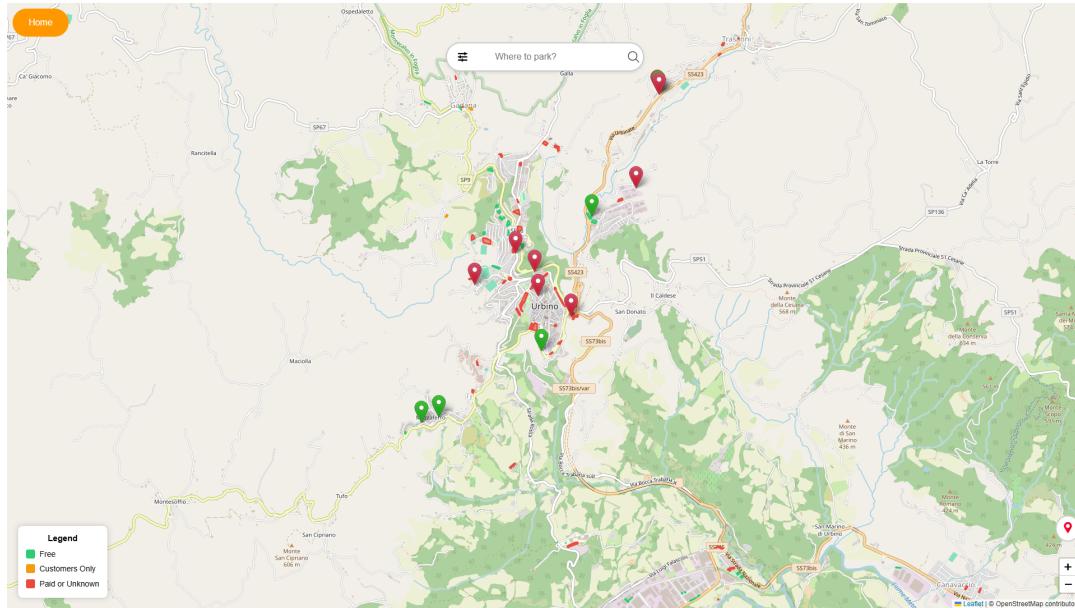


Figure 8: Pagina per visualizzare la mappa con tutti parcheggi

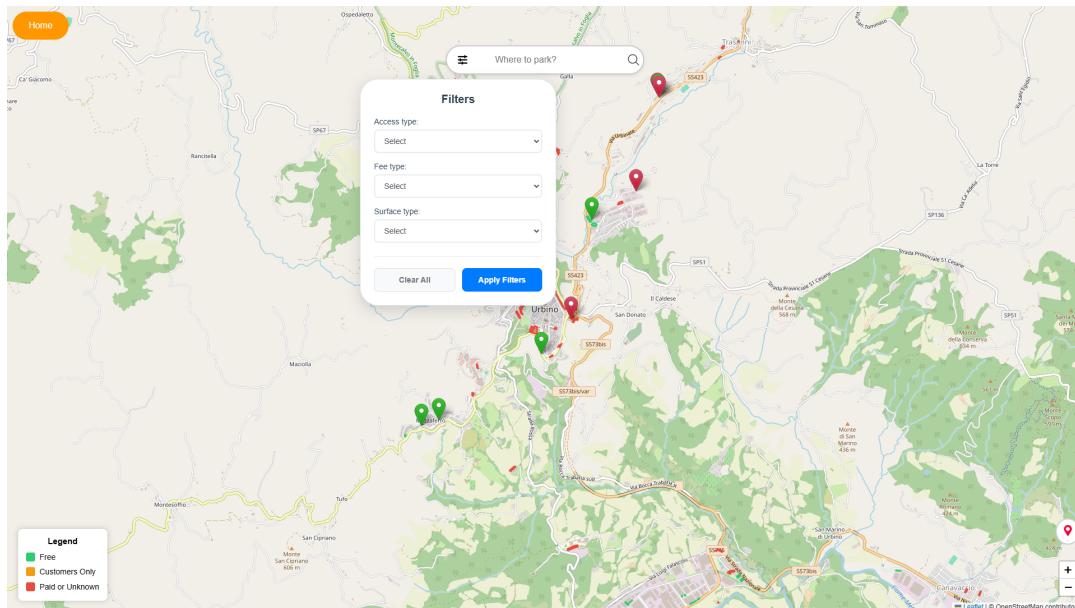


Figure 9: Pannello dei Filtri

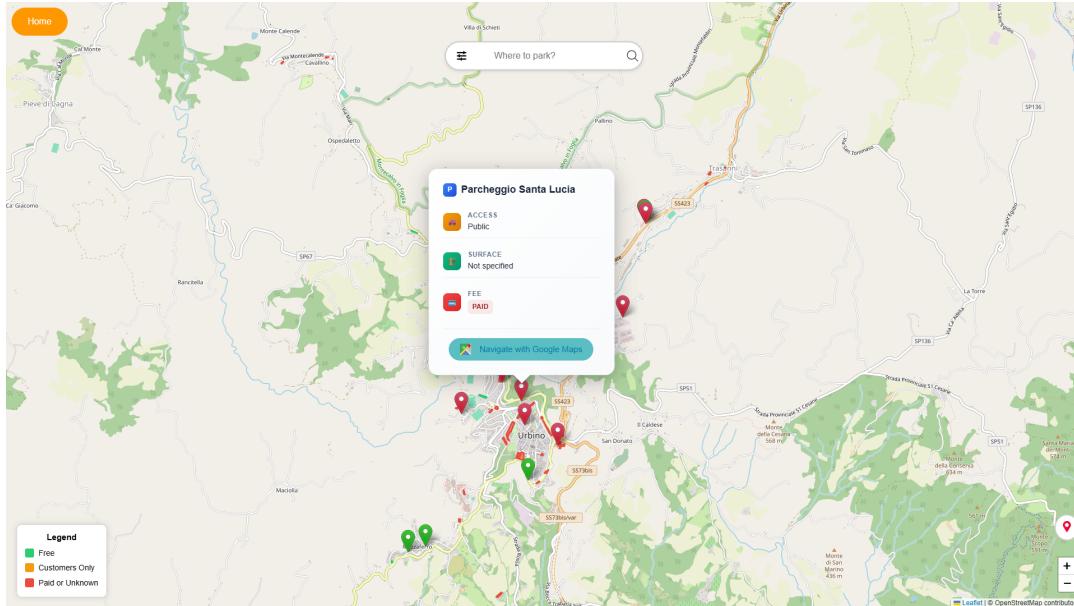


Figure 10: PopUp che mostra le informazioni del parcheggio

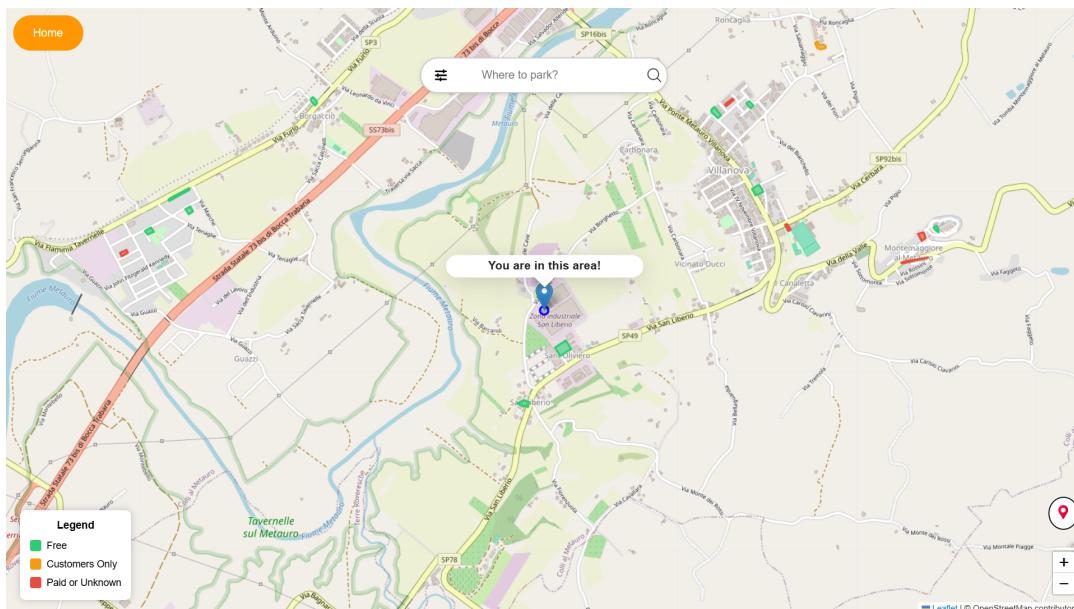


Figure 11: Geolocalizzazione

5.3 Pagina per Aggiungere un Parcheggio

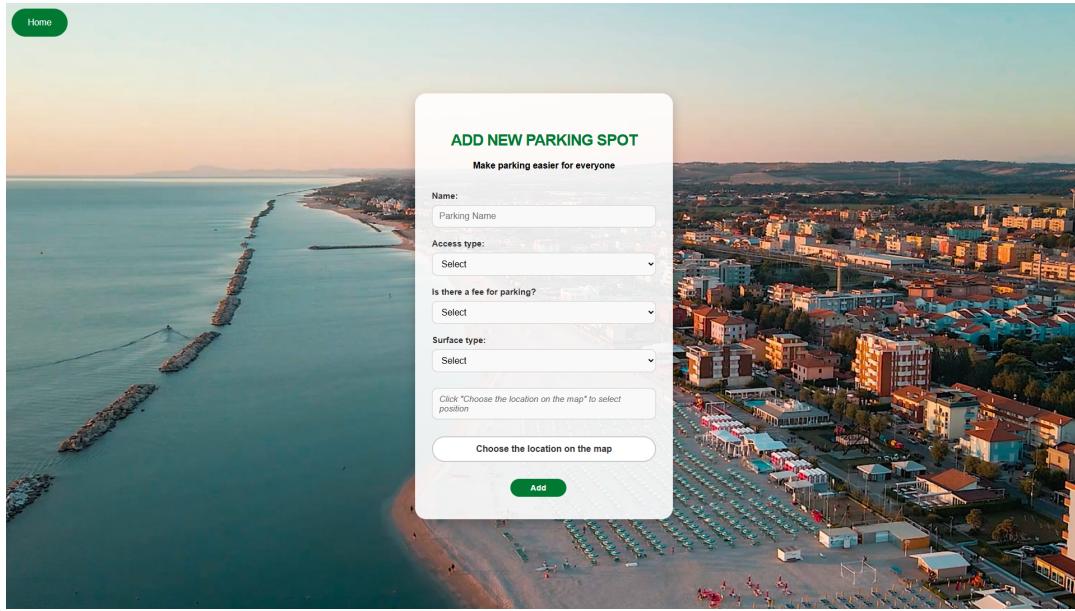


Figure 12: Pagina per aggiungere un nuovo parcheggio

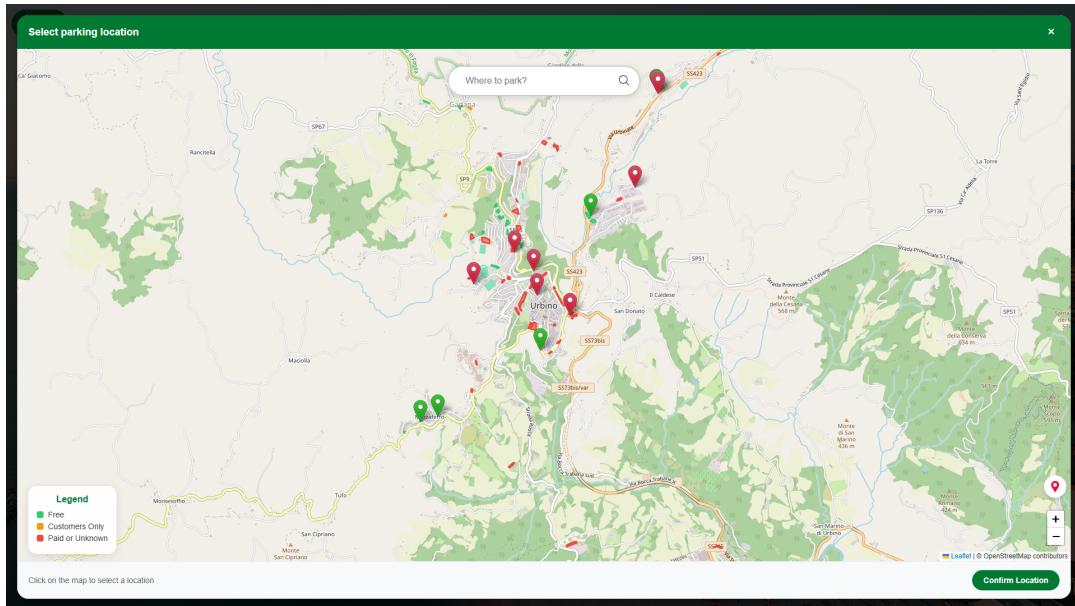


Figure 13: Pagina per selezionare la posizione del nuovo parcheggio

5.4 Pagina per Rimuovere un Parcheggio

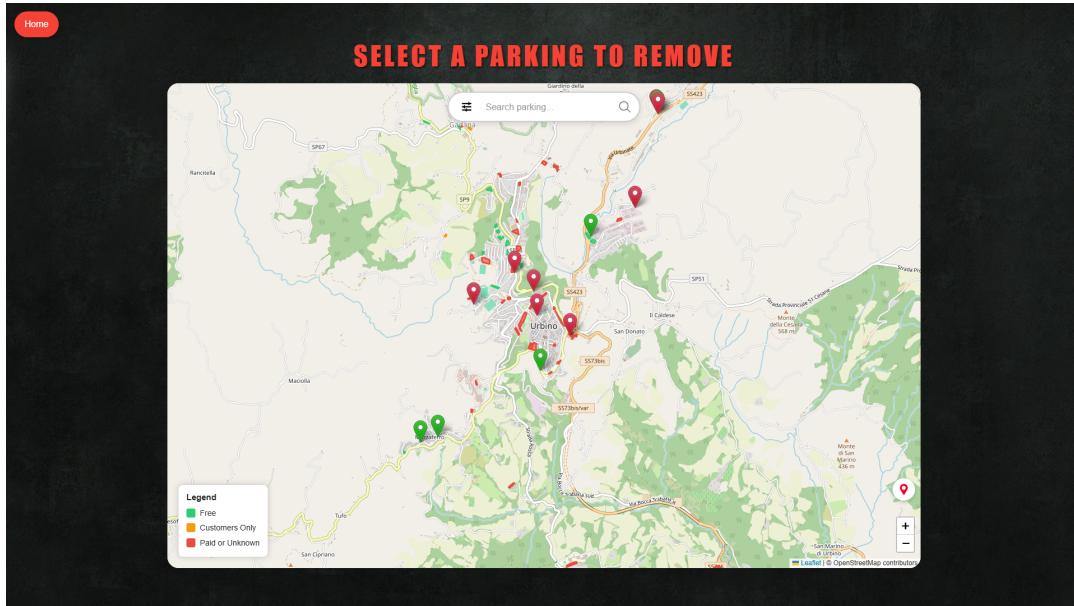


Figure 14: Pagina per la rimozione dei parcheggi

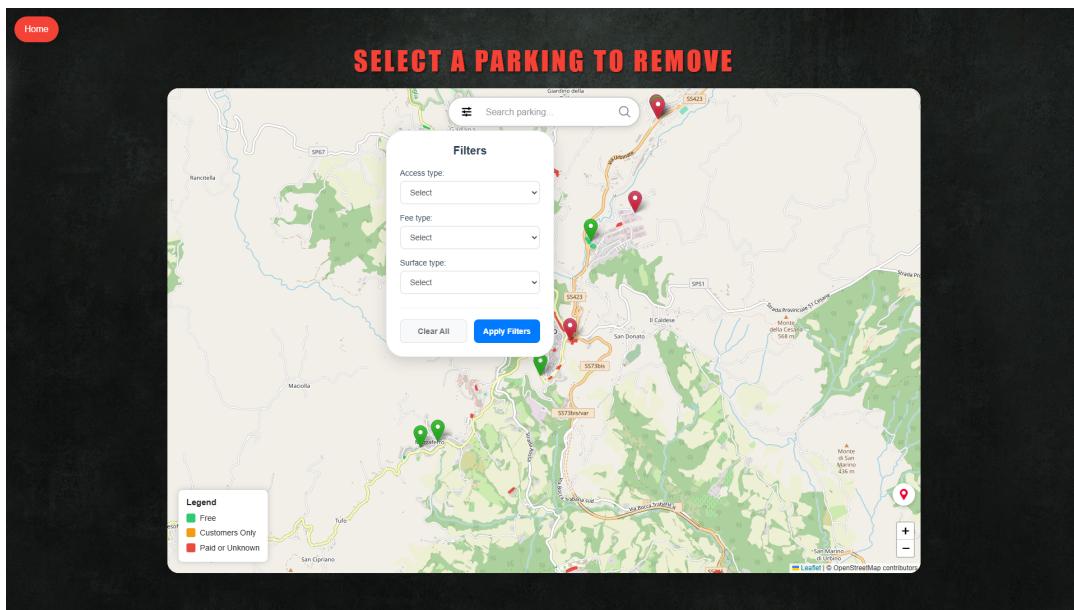


Figure 15: Pannello dei Filtri

5.5 Pagina per Modificare un Parcheggio

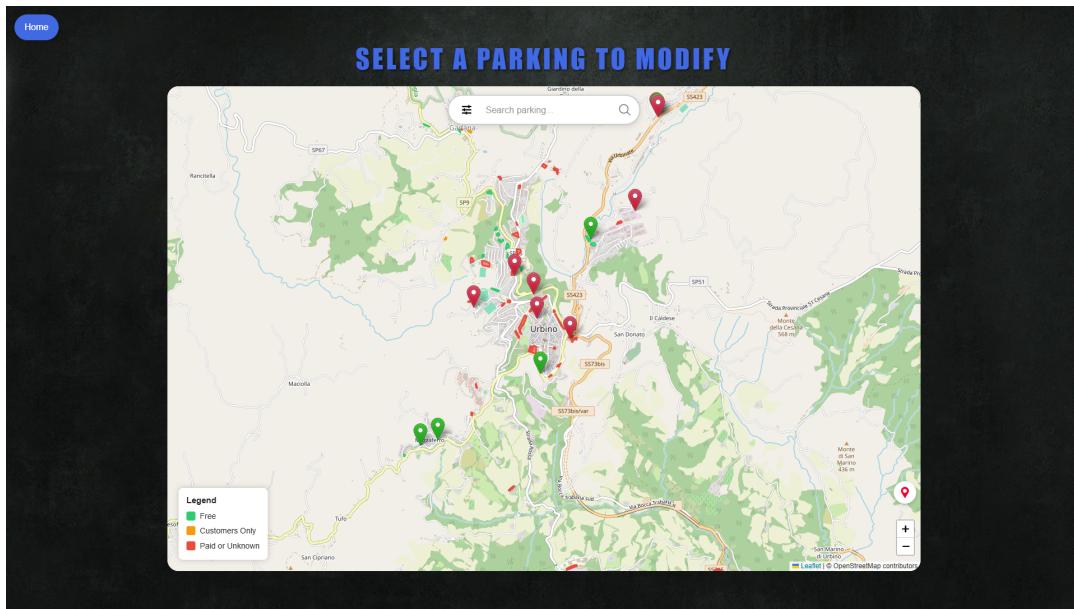


Figure 16: Pagina per modificare i dati di un parcheggio

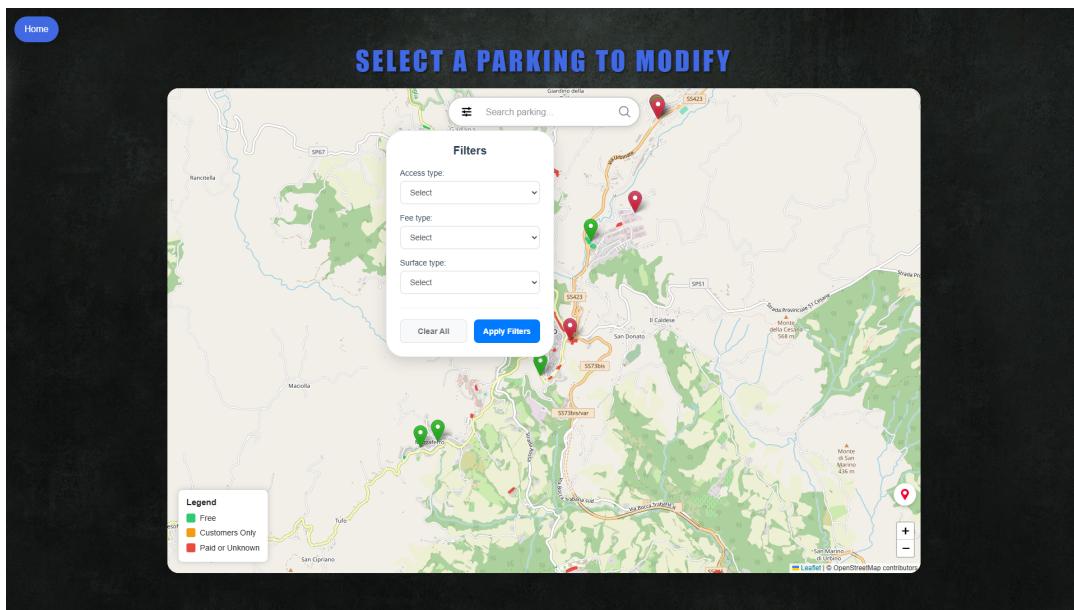


Figure 17: Pannello dei Filtri

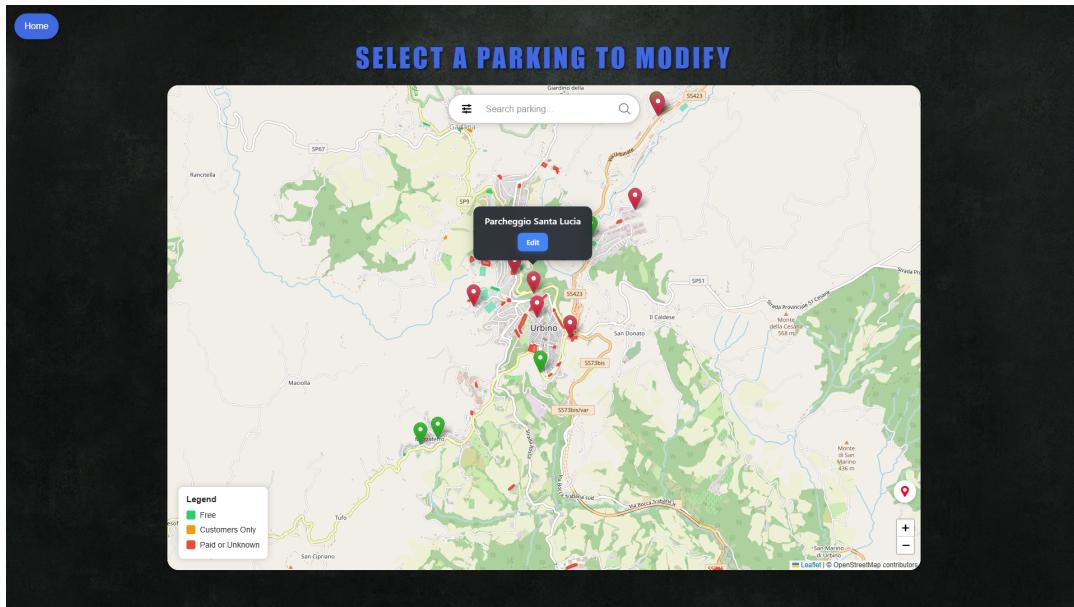


Figure 18: PopUp per selezionare il parcheggio da modificare

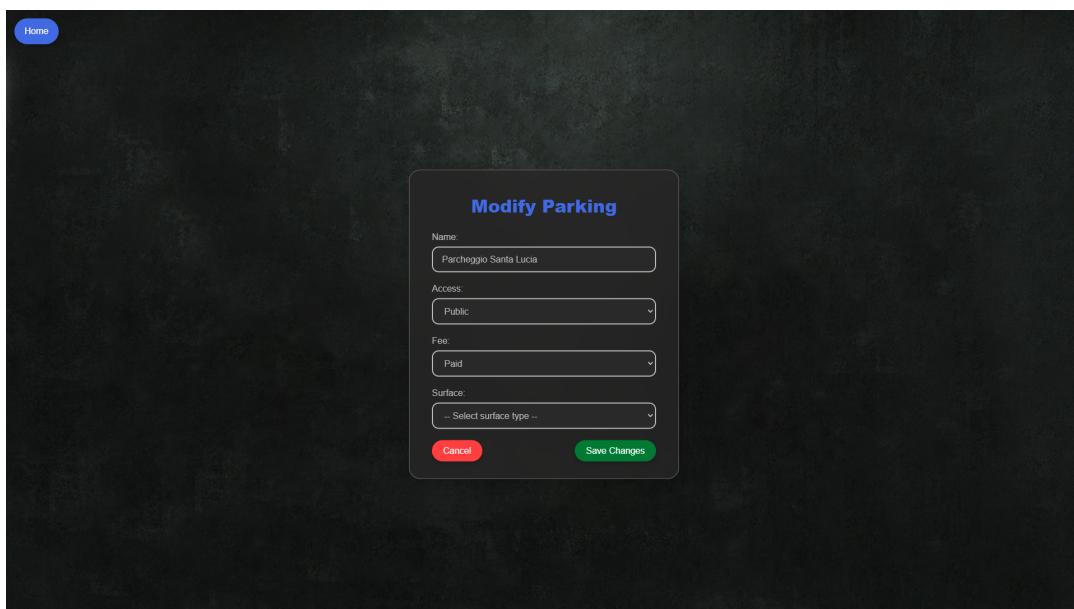


Figure 19: Form per modificare gli attributi di un parcheggio