UNIVERSITÀ DEGLI STUDI DI URBINO CARLO BO

Corso di Laurea in Informatica Applicata

Anno Accademico 2024/2025

Mappa dei Siti Contaminati della Regione Emilia-Romagna

Progetto d'esame per il corso di Tecnologie Web per la Gestione del Territorio

Milena Balducci, matricola 321791

Indice

1 Descrizione del servizio implementato

Il servizio web realizzato ha lo scopo di permettere all'utente di visualizzare, aggiungere, modificare o rimuovere i siti contaminati presenti nella regione Emilia-Romagna.
Utilizza i dati **Open Data** forniti dalla Regione e li presenta tramite una mappa interattiva, consentendo di esplorarli in modo intuitivo e dinamico.
La mappa è arricchita da funzionalità di ricerca tramite codice univoco e dalla possibilità di filtrare i siti in base a diversi parametri, oltre a opzioni di visualizzazione dei dettagli per ciascun sito.

Tramite l'interfaccia utente, l'applicazione web permette all'utente di:

- visualizzare i siti contaminati sulla mappa
- filtrare i siti in base a provincia, comune o a informazioni avanzate come lo stato della bonifica e/o della messa in sicurezza; o di cercarli tramite il codice identificativo
- accedere a informazioni dettagliate sui singoli siti
- modificare i dati relativi ai siti
- aggiungere o rimuovere un sito.

Il servizio è stato pensato per fornire un'interfaccia intuitiva e responsiva, adatta sia a utenti tecnici (come operatori della pubblica amministrazione o ARPA) sia a cittadini interessati allo stato ambientale del territorio.
L'obiettivo è quello di fornire un meccanismo per accedere in maniera semplice e aperta a tutti ai dati ambientali riguardanti la regione.

2 Architettura e Scelte Implementative

2.1 Componenti software

L'architettura del sistema, di tipo *client-server*, si compone di due parti principali:

- un **backend** realizzato in **Node.js** con **Express**, che gestisce la lettura e il filtraggio dei dati da file JSON e risponde alle richieste HTTP
- un **frontend** in **HTML, CSS** e **JavaScript**, che si occupa della visualizzazione dei dati, dell'interazione con l'utente e dell'invio delle richieste HTTP al server.

La comunicazione tra client e server avviene tramite HTTP, utilizzando il metodo **fetch** per richiedere e inviare dati.

Il backend espone un insieme di servizi **RESTful** che permettono di leggere, filtrare, aggiungere, modificare o eliminare informazioni relative ai siti contaminati. I dati vengono caricati da un file JSON e serviti in formato JSON.

L'intero servizio è pensato per funzionare in locale o in rete interna, ed è facilmente estendibile per operare su database o integrare funzioni di autenticazione e gestione avanzata degli utenti.

2.2 Interfaccia utente

Il frontend è stato implementato tramite una **single-page web app** sviluppata in HTML, CSS e JavaScript, e utilizza tecnologie come:

- **Fetch API** per la comunicazione asincrona col backend;
- **Leaflet.js** per la visualizzazione dei dati su una mappa interattiva;
- **CSS Grid e Flexbox** per il layout responsivo.

I file statici (HTML, CSS, JS) sono posizionati nella cartella *public*, che Express rende accessibile.

Il flusso tipico prevede che l'utente, al caricamento della pagina, interagisca con l'interfaccia web (filtri, bottoni, mappa), che a sua volta effettua chiamate HTTP al backend per ottenere o aggiornare i dati. Le risposte vengono mostrate dinamicamente senza necessità di ricaricare la pagina.

L'interfaccia utente è suddivisa in sezioni visibili alternativamente:

- **default:** la sezione caricata automaticamente all'avvio, mostra il numero totale dei siti registrati, quello dei siti bonificati e dei siti messi in sicurezza
- **cerca:** permette di cercare un sito tramite il suo codice identificativo
- **filtri:** permette di filtrare i siti per provincia, comune, stato della bonifica ecc.

- **aggiungi sito:** mostra un form che permette di aggiungere un nuovo sito contaminato alla mappa
- **infopanel:** mostra informazioni aggiuntive relative a un sito, come lo stato della bonifica e della messa in sicurezza ed eventuali note
- **modifica sito:** permette di modificare le informazioni aggiuntive (informazioni come provincia, comune, indirizzo e coordinate si ritengono definitive e non modificabili).

La visibilità delle sezioni è gestita tramite Javascript: a ciascuna sezione corrisponde un **id** e un button (collocati in punti diversi della pagina, a seconda della sezione), ciascuno dei quali chiama una funzione `mostraSezione(id)` che assegna la classe attiva alla sezione che si è scelto di visualizzare, rimuovendola da quella precedentemente visibile. Al caricamento della pagina, la prima sezione visibile è **default**.

La mappa interattiva è stata posta sotto le sezioni che compono l'interfaccia utente. Essa riporta i marker di ciascuno dei siti contaminati registrati: cliccando su ciascun marker si apre un pop-up tramite il quale è possibile visualizzare le informazioni base relative al sito selezionato (come codice, indirizzo, comune e provincia ecc.), mentre per visualizzare le informazioni aggiunte e modificare o eliminare il sito è sufficiente cliccare sul button posto nel pop-up.

2.2.1 CSS: Flexbox e Grid

Flexbox e **Grid** rappresentano l'evoluzione moderna del layout in CSS, offrendo un controllo molto più preciso e leggibile sulla disposizione degli elementi rispetto al sistema basato su float.

Nell'ambito di questo progetto il layout generale delle sezioni, ciascuna divisa in un header e un elemento di classe `card`, è stato sviluppato usando flexbox e la sua proprietà `flex-direction: column` (più raramente si è fatto uso di `row`), mentre grid è stato utilizzato principalmente per il layout dei form e l'allineamento dei `select` nella sezione filtri.

Flexbox e Grid non solo presentano una maggiore semplicità di utilizzo ma rendono il sistema più facilmente mantenibile, per questo motivo si è deciso di preferirli al layout basato su float, che risulta più verboso e difficile da modificare in seguito.

2.2.2 Leaflet

Leaflet è una libreria JavaScript open source leggera, semplice da usare e altamente estendibile, ideale per applicazioni web che richiedono mappe interattive.
A differenza di soluzioni più complesse e vincolate da costi o licenze (come **Google Maps**), Leaflet consente di integrare facilmente layer personalizzati, marker, popup e controlli, senza dipendere da API proprietarie. Inoltre, offre un ottimo supporto per **OpenStreetMap**, il che la rende perfetta per progetti che richiedono flessibilità e sostenibilità a lungo termine, o che presentano vincoli di costo.

2.3 Comunicazione client-server

Come menzionato in precedenza, la comunicazione avviene tramite **fetch API** dal lato frontend, utilizzando chiamate HTTP verso il server Express.
Al caricamento della pagina, i dati sono caricati in memoria dal file JSON tramite `GET`.
I filtri vengono applicati lato server su richiesta dell'utente (es. filtro per provincia), e l'applicazione di ogni filtro triggera una chiamata all'API con metodo `GET`.
Altre chiamate dello stesso tipo vengono effettuate dopo ogni `POST`, `PUT` e `DELETE`, in modo da mantenere aggiornata la lista dei siti.

2.4 Caricamento e salvataggio dei dati

2.4.1 Formato dei dati

Tutti i dati sono scambiati in formato JSON. Ogni sito è rappresentato da un oggetto con i seguenti campi principali:

- **codice:** identificativo univoco del sito
- **comune, provincia, indirizzo, attività:** dati anagrafici del sito
- **messa_sicurezza_emergenza, messa_sicurezza_operativa, messa_sicurezza_permanenete, bonifica, bonifica_sicurezza:** fasi della bonifica
- **procedura, note:** informazioni aggiuntive
- **lat, lon:** coordinate geografiche

Tuttavia, il file originale dal quale vengono parsati i dati presenta un formato delle chiavi differente da quello sopracitato. Per questo motivo, dopo il caricamento è necessario convertire le chiavi per fare in modo che combacino con quelle attese dal Web Service.
Di seguito un piccolo estratto del dataset che mostra il formato originale dei dati:

```
{
   "Codice": "pc000026",
   "Comune": "BESENZONE",
   "Provincia": "PIACENZA",
   "Indirizzo": "strada del bersano,",
   "Attivita": "industriale",
   "Messa in sicurezza d'emergenza": "No",
   "Messa in sicurezza operativa": "No",
   "Messa in sicurezza permanente": "No",
   "Bonifica e ripristino ambientale": "Si",
   "Bonifica e ripristino ambientale con misure di sicurezza":
       "No",
   "Procedura": "non art. 242 non bonificato",
   "Note": "",
   "Latitudine": 44.986569,
   "Longitudine": 9.952705
},
```

Le chiavi relative alla bonifica e alla messa in sicurezza dei siti risultano molto lunghe, oltre a presentare spazi, per questo motivo si è scelto di modificarle. Per la conversione delle chiavi si fa uso di un oggetto `campiSito`:

```
const campiSito = {
  codice: "Codice",
  comune: "Comune",
  provincia: "Provincia",
  indirizzo: "Indirizzo",
  attivita: "Attivita",
  messa_sicurezza_emergenza: "Messa in sicurezza d'emergenza",
  messa_sicurezza_operativa: "Messa in sicurezza operativa",
  messa_sicurezza_permanenete: "Messa in sicurezza permanente
     ",
  bonifica: "Bonifica e ripristino ambientale",
  bonifica_sicurezza: "Bonifica e ripristino ambientale con
     misure di sicurezza",
  procedura: "Procedura",
  note: "Note",
  lat: "Latitudine",
  lon: "Longitudine"
};
```

2.4.2 Lettura e Scrittura

Per la lettura/scrittura dei dati, il backend fa uso delle librerie:

- **fs** per la lettura/scrittura su file, tramite `readFileSync` e `writeFile`, con codifica `utf-8`,
- **JSON** per il parsing dei dati tramite `stringify` e `parse`,
- **path** per la creazione delle variabili che contengono i filepath dei due dataset utilizzati.

I dati vengono letti dal file JSON, che si trova nella cartella *server/data*, tramite `readFileSync`. Dopo la lettura, si costruisce un nuovo oggetto `site` mappando ciascun oggetto JSON per sostituire le chiavi originali con quelle contenute in `campiSito`.

```
// Caricamento dati
function loadSitesFromJSON() {
  const jsonData = fs.readFileSync(jsonFilePath, "utf-8");
  const rawData = JSON.parse(jsonData);

  return rawData.map(entry => {
    const site = {};

    for (const [key, originalKey] of Object.entries(campoMappa
       )) {
      site[key] = (key === "lat" || key === "lon")
        ? parseFloat(entry[originalKey])
        : entry[originalKey];
```

```
    }

    return site;
  });
}
```

Il processo di scrittura è simile: l'oggetto JSON passato alla funzione `saveData` viene mappato per riconvertire le chiavi in quelle presenti nel file Open Data originale, in modo da mantenere la coerenza tra i vari oggetti presenti nel file ed evitare bug ed errori indesiderati, e poi salvato tramite `writeFile`.

```
// Scrittura dati
function saveData(data, jsonFilePath) {
  const mappedData = data.map(entry => {
    const mapped = {};
    for (const [key, originalKey] of Object.entries(campoMappa
       )) {
      mapped[originalKey] = entry[key];
    }
    return mapped;
  });

  fs.writeFile(jsonFilePath, JSON.stringify(mappedData, null,
     2), err => {
    if (err) {
      console.error("Errore nel salvataggio:", err);
    } else {
      console.log("File salvato correttamente!");
    }
  });
}
```

3 Documentazione dell'API implementata

3.1 API GET

URL: `/comuni`.
Restituisce l'elenco dei comuni presenti nella regione, divisi per provincia.

- **Dati in input:** nessuno
- **Dati in output:** array di JSON
- **Status risposta:** `200 OK`
- **Codici di errore gestiti:**
  - `500 Internal Server Error`

URL: `/comuni?provincia=p`.
Restituisce l'elenco dei comuni presenti nella provincia indicata.

- **Dati in input:** parametro provincia (string).
- **Dati in output:** JSON
- **Status risposta:** `200 OK`
- **Codici di errore gestiti:**
  - `500 Internal Server Error`

Esempio: `GET /comuni?provincia=RIMINI` restituisce l'elenco dei comuni della provincia di Rimini.

URL: `/siti`.
Restituisce l'intero elenco dei siti contaminati.

- **Dati in input:** nessuno
- **Dati in output:** array di JSON
- **Status risposta:** `200 OK`
- **Codici di errore gestiti:**
  - `500 Internal Server Error`

URL: `/siti?querystring`.
Restituisce l'intero elenco dei siti contaminati che corrispondono ai filtri in `querystring`.

- **Dati in input:** parametri passati tramite querystring
- **Dati in output:** array di JSON
- **Status risposta:** `200 OK`
- **Codici di errore gestiti:**
  - `500 Internal Server Error`

Esempio: `GET /siti?provincia=RIMINI` restituisce tutti i siti della provincia di Rimini.

3.2 API POST

URL: `/siti`.
Aggiunge un nuovo sito contaminato alla mappa.

- **Dati in input:** oggetto JSON sotto forma di stringa.
- **Dati in output:** nessuno
- **Status risposta:** `201 CREATED`
- **Codici di errore gestiti:**
  - `500 Internal Server Error`
  - `409 Conflict`

Esempio:

```
POST /siti
Body:
 {"provincia":"CITTA' METROPOLITANA DI BOLOGNA","comune":"
    CASTEL SAN PIETRO TERME","indirizzo":...}
```

3.3 API PUT

URL: `/siti/:codice`.
Aggiorna i dati di un sito esistente, identificato dal codice.

- **Dati in input:** il codice del sito da aggiornare e la stringa corrispondente all'oggetto JSON contenente i dati del sito da modificare.
- **Dati in output:** nessuno
- **Status risposta:** `200 OK`
- **Codici di errore gestiti:**
  - `404 Not Found`
  - `500 Internal Server Error`

Esempio:

```
PUT /siti/pc000051
Body:
 {codice: 'pc000051', comune: 'PIACENZA', provincia: 'PIACENZA
    ', indirizzo: 'viale Malta,11', attivita: 'industriale
    ',...}
```

3.4 API DELETE

URL: `/siti/:codice`.
Elimina il sito corrispondente al codice fornito.

- **Dati in input:** il codice del sito da eliminare
- **Dati in output:** nessuno
- **Status risposta:** `200 OK`
- **Codici di errore gestiti:**
  - `404 Not Found`
  - `500 Internal Server Error`

Esempio: `DELETE /siti/FE000013` elimina il sito con codice FE000013.

4 Esempio di utilizzo del servizio

4.1 Test da terminale

Come primo test, se è deciso di verificare il funzionamento dell'API da terminale, tramite il comando `curl`, per verificare che il backend risponda correttamente alle richieste effettuamente a prescindere dall'intervento dell'interfaccia grafica. Di seguito alcuni esempi di richieste inviate, incluse alcune con dati volutamente errati per testare gli status restituiti.

4.1.1 GET

Corretta: si caricano tutti i siti presenti nel dataset.

```
curl -i http://localhost:3000/siti

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 143681
ETag: W/"23141-09cdX1OFIg5kIQB75OFKExFhpLE"
Date: Thu, 19 Jun 2025 19:45:54 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"success":true,"message":"Tutti i siti restituiti con
   successo","results":[{"codice":"pc000026","comune":"
   BESENZONE","provincia":"PIACENZA","indirizzo":"strada del
   bersano,","attivita":"industriale","
   messa_sicurezza_emergenza":"No",...}]
```

Corretta: si cerca un sito con un codice inesitente. Questa chiamata non solleva errori, in quanto è possibile che non esista un sito che corrisponde ai filtri selezionati (incluso il codice), quindi il server risponde correttamente con `200 OK`.

```
curl -i http://localhost:3000/siti?codice=57dhfh8

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 89
ETag: W/"59-h1KauKDFynlHUAGLHYy1wfUDviA"
Date: Thu, 19 Jun 2025 20:18:40 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"success":true,"message":"Nessun sito trovato corrispondente
   ai parametri","results":[]}
```

4.1.2 POST

Errata: in questo esempio si è provato a inserire un sito con un codice univoco già presente nel dataset: come da aspettative, il server risponde con `409 Conflict`.

```
curl -i -X POST http://localhost:3000/siti -H "Content-Type:
   application/json" -d '{"provincia":"RIMINI","comune":"
   SALUDECIO","indirizzo":"Via Saporetti 1","attivita":"
   nessuna","codice":"09SA11245","Longitudine":"43.10854","
   Latitudine":"12.39489","messa_sicurezza_emergenza":"No","
   messa_sicurezza_operativa":"No","messa_sicurezza_permanente
   ":"No","bonifica":"No","bonifica_sicurezza":"No","procedura
   ":"non art. 242 non bonificato","note":""}'

HTTP/1.1 409 Conflict
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 49
ETag: W/"31-yFbeLFlHKfIshPjNfiBPjDi5PlU"
Date: Thu, 19 Jun 2025 19:53:11 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"error":"Sito gi  esistente con questo codice"}
```

4.1.3 PUT

Corretta: il codice selezionato è presente nel dataset, la modifica avviene correttamente.

```
curl -i -X PUT http://localhost:3000/siti/pc000051 -H "Content
   -Type: application/json" -d '{"attivit ": "industriale
   ",...}'

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 646
ETag: W/"286-ZNvgkeU3q4kvb9AlCcRw+Fnm+Ow"
Date: Thu, 19 Jun 2025 20:09:55 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"success":true,"message":"Sito aggiornato","sito":{"attivit
   ":"industriale","bonifica":"No","bonifica_sicurezza":"Si","
   codice":"pc000051","comune":"PIACENZA","indirizzo":...}
```

Errata: inserendo invece un codice errato, la PUT risponde con `404 Not Found`.

```
curl -i -X PUT http://localhost:3000/siti/pc000051H -H "
   Content-Type: application/json" -d '{"attivit ": "
   industriale",...}'
```

```
HTTP/1.1 404 Not Found
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 28
ETag: W/"1c-ZlB/5WHcuWv8eJAvwDiq1MCkHh0"
Date: Thu, 19 Jun 2025 21:18:19 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"error":"Sito non trovato"}
```

4.1.4 DELETE

Corretta: si elimina un sito con un codice univoco effettivamente presente nel dataset. Si ottiene il risultato atteso, ovvero `200 OK`.

```
curl -i -X DELETE http://localhost:3000/siti/200495

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 61
ETag: W/"3d-nj/ftIY50bMu3NRcQlPA5raVuJ8"
Date: Thu, 19 Jun 2025 19:55:09 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"success":true,"message":"Sito con codice 200495 eliminato"}
```

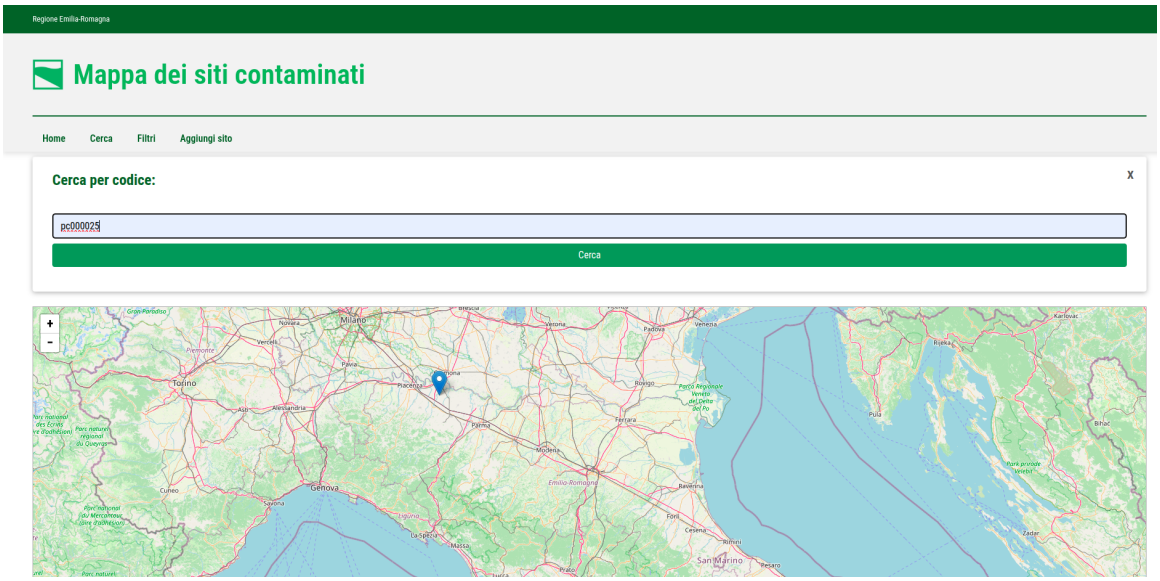
Errata: in questo caso invece si è tentato di eliminare un sito con un codice inesistente. Il server risponde correttamente con `404 Not Found`.

```
curl -i -X DELETE http://localhost:3000/siti/200495

HTTP/1.1 404 Not Found
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 28
ETag: W/"1c-ZlB/5WHcuWv8eJAvwDiq1MCkHh0"
Date: Thu, 19 Jun 2025 19:55:13 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"error":"Sito non trovato"}
```

4.2 Test da interfaccia utente

4.2.1 Ricerca per codice

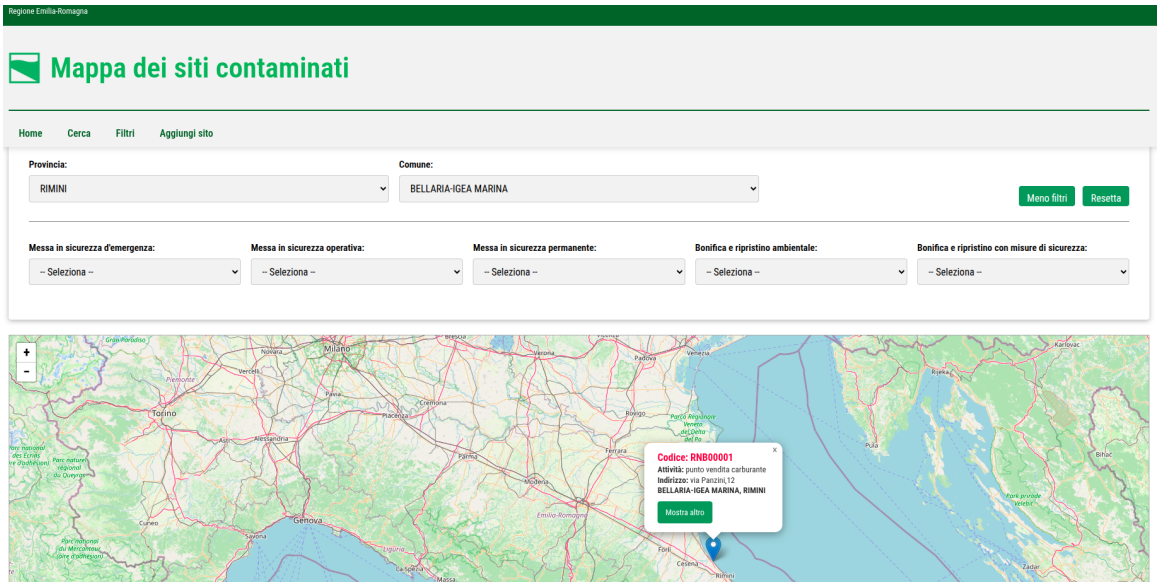
In questo esempio, si è effettuata la ricerca del sito con codice pc000025.



Come è possibile vedere dall'immagine sottostante, il backend risponde correttamente alla richiesta del frontend, restituendo il sito desiderato.

4.2.2 Applicazione filtri

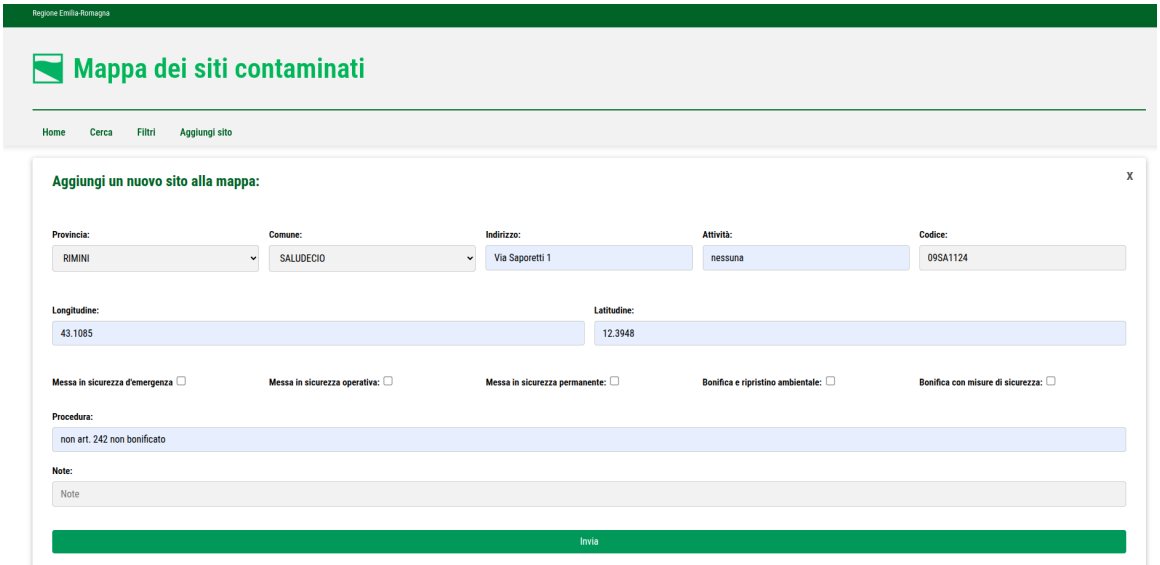
Ciascuna selezione di un filtro invia una richiesta `GET` al backend che ha come querystring la coppia chiave-valore del filtro selezionato, che viene poi salvata per essere eventualmente concatenata con altri filtri. Quindi, vista la configurazione nell'immagine sottostante ci si aspetta che l'URL abbia una querystring con due campi, uno per provincia e uno per comune.



L'immagine seguente mostra la corretta formattazione sia della request che della response.

4.2.3 Aggiunta di un nuovo sito

In questo paragrafo si dimostra l'aggiunta di un nuovo sito alla mappa tramite l'apposito form presente nell'interfaccia grafica.



Anche in questo caso, come nel test da terminale, il server risponde nel modo atteso (`200 OK`) e aggiunge correttamente il sito alla mappa.

5 Conclusioni

Come dimostrato nella sezione precedente, il servizio web presentato risponde correttamente alle richieste inviate dall'utente sia da interfaccia grafica sia da terminale, restituendo gli stati e i risultati attesi.
L'interfaccia, grazie alla facilità di navigazione e utilizzo, consente di accedere ai dati ambientali in maniera intuitiva e veloce.

In futuro, il servizio potrebbe prestarsi a modifiche o espansioni, per esempio tramite l'utilizzo di database che vadano a sostituire i file JSON, o aggiungendo delle restrizioni sulle azioni che è possibile effettuare sui siti (ad esempio permettendone l'eliminazione e la modifica solo al personale tecnico-amministrativo).

6 Fonti Dati e Servizi Esterni

- **Open Data siti contaminati:** http://www.datiopen.it/it/opendata/Regione_Emilia_Romagna_Siti_conta
- **Dataset comuni per provincia:** creato manualmente utilizzato gli elenchi dei comuni per provincia disponibili online
- **API Leaflet:** https://leafletjs.com/reference.html
- **Font utilizzato per il design del frontend:**https://fonts.google.com/