Progetto di Tecnologie Web per la Gestione del Territorio

Samuele Caporale – 329096

2024/2025

Indice

1 Descrizione del servizio **3**

2 Tecnologie utilizzate **4**
2.1 Frontend 4
2.2 Backend 4

3 Modello di processo **5**
3.1 Versioning 5
3.1.1 Major 5
3.1.2 Minor 5
3.1.3 Patch 5

4 Architettura del servizio **6**
4.1 Architettura 6
4.2 Modello di comunicazione 6
4.3 Scelte Implementative 6
4.3.1 Directory 6
4.4 Client 6
4.4.1 Public 6
4.4.2 Assets 7
4.4.3 Script 7
4.4.4 Service 7
4.5 Server 8
4.5.1 Utils 8
4.5.2 Upload 8

5 Dati **9**
5.1 Open Data 9
5.2 Licenza 9

6 Documentazione API **10**
6.1 Endpoints 10
6.2 Formato delle richieste 11
6.2.1 Caricamento file 11
6.2.2 Aggiunta misurazione a file esistente 12
6.2.3 Ottenimento file specifico 13
6.2.4 Ottenimento lista file 14
6.2.5 Eliminazione file 15
6.2.6 Eliminazione misurazione specifica 15
6.2.7 Modifica misurazione specifica 16

7 Testing End-to-End **17**

1 Descrizione del servizio

AirScope è un web service per il monitoraggio del territorio nell'ambito "qualità dell'aria" che consente:

- Caricamento file in formato csv
- Aggiunta di una nuova misurazione
- Modifica della misurazione
- Cancellazione di una misurazione
- Cancellazione file
- Visualizzazione dei valori medi
- Visualizzazione dei file caricati
- Visualizzazione tabellare delle misurazioni
- Visualizzazione a infografica delle misurazioni

Il sistema è basato su misurazioni provenienti da open data e pensato per la regione Marche. Si utilizzeranno come riferimento i dati ARPA della città di Urbino, nonostante ciò il sistema può essere utilizzato per ogni città qualora i file caricati rispettino lo standard del modello di esempio mostrato.

2 Tecnologie utilizzate

2.1 Frontend

- **HTML5, CSS3 e javaScript**
  Vengono utilizzate lato client rispettivamente per la creazione delle pagine e dei contenuti, stile grafico personalizzato, gestione dinamica dei contenuti e feedback.
- **Bootstrap 5**
  Scelto e utilizzato per la creazione della pagine sfruttando la struttura a colonne e i componenti come Toast, Modal e le Icon.
- **Chart.js**
  Libreria scelta per la rappresentazione a infografica degli open data.
- **jQuery**
  Libreria js adottata per la retro compatibilità tra browser, chiamate ajax e laddove possibile semplificare la sintassi "Write less, do more".
- **LocalStorage**
  Tecnologia di web storage nativa del browser per memorizzare dati sotto forma di chiave e valore, utilizzata per memorizzare il nome del file attualmente in uso.

2.2 Backend

- **Node.js**
  Scelto come ambiente d'esecuzione javaScript lato server a run-time e per la gestione dei pacchetti con npm.
- **Express.js**
  Framework utilizzato per creare il web server, definire le route e gestire le chiamate HTTP.
- **Multer**
  Middleware open source usato per l'upload di file da form.
- **csv-parse**
  Libreria utilizzata per fare il parsing dei file csv caricati.
- **fs**
  Modulo di node utilizzato per fare operazioni sul file system.
- **path**
  Modulo di node utilizzato per lavorare con i path in modo da renderlo indipendente dal sistema operativo.
- **file json**
  JavaScrit Object Notation è la struttura dati scelta per salvare in modo persistente le misurazioni e la lista dei file caricati.

3 Modello di processo

Per lo sviluppo è stato adottato un approccio Agile combinato al Modello incrementale. Refactoring costante, testing in parallelo all'implementazione tramite la console hanno caratterizzato lo sviluppo di tutto il progetto.

- Nelle prime fasi del progetto è stata raggiunta una versione minima funzionante.
- A fine di ogni incremento è stato prodotto un Backlog cartaceo contenente le modifiche e i bug da fissare per la fase successiva.
- Inoltre è stata tenuta traccia dello sviluppo tramite Git adottando la politica di versioning (Major.Minor.Patch) come segue:

3.1 Versioning

3.1.1 Major

Gli incrementi Major sono legati a modifiche strutturali non retro compatibili, come l'aggiunta del Local Storage per tenere traccia del file su cui operare.

3.1.2 Minor

Gli incrementi Minor sono quelli che introducono nuove funzionalità compatibili, come l'aggiunta di un'endpoint o di un nuovo componente della GUI.

3.1.3 Patch

Gli incrementi Patch sono relativi a correzioni di bug o a piccole modifiche.

4 Architettura del servizio

4.1 Architettura

L'architettura del servizio adottata è la Client-Server

4.2 Modello di comunicazione

Il modello di comunicazione è Request-Response tramite protocollo HTTP.
Il web server risponde alle richieste del Client tramite il modello RESTful utilizzando i verbi CRUD.

4.3 Scelte Implementative

4.3.1 Directory

Le directory sono state suddivise come segue:

```
/ (root)
├── node_modules/
├── public/
│   ├── assets/
│   │   ├── img/
│   │   ├── svg/
│   │   ├── script/
│   │   └── style.css
│   ├── service/
│   │   ├── dashboard.html
│   │   ├── analytics.html
│   │   └── request.html
│   ├── index.html
│   └── service.html
├── upload/
├── utils/
└── server.js
```

4.4 Client

4.4.1 Public

Lato Client ciò che è pubblico si trova dentro la directory 'public' eccetto le librerie aggiunte tramite npm (bootstrap, chart.js, jQuery) che vengono esposte dal server.
Alla radice di public troviamo la 'index.html' che funge da pagina d'atterraggio per accedere al servizio e la pagina 'service.html' che contiene la GUI di base che comprende: menu, sezione principale del contenuto, zona Modal e la zona riservata ai Toast per il feedback.

- È stato utilizzato un foglio di stile personalizzato per compensare la rigidità di Bootstrap e definire alcuni stili generali per i componenti, come ad esempio il 'general-container' per ogni contenuto mostrato nella sezione principale.

4.4.2 Assets

Questa contiene tutte le risorse statiche per le pagine del servizio, lo stile personalizzato, gli script per popolare, manipolare ed ascoltare l'HTML e gli eventi ad esso collegati.

- La validazione all'aggiunta della singola misurazione avviene direttamente dal form html con gli appositi attributi.
- È stata lasciata la possibilità di omettere valori degli inquinanti in fase di modifica e inserimento open data dal momento che in certune situazioni i sistemi di rilevamento possono essere spenti o non avere tutti i dati disponibili.

4.4.3 Script

La directory Script contenuta in Assets contiene il file principale 'main.js', questo si occupa di gestire il caricamento del contenuto della pagina, chiamando tutte le funzioni che popolano tabelle, Chart, Toast e il calcolo dei valori medi.

- È stato deciso di utilizzare una variabile 'debug' per attivare e disattivare tutti i log scritti durante l'implementazione.
- È stato deciso anche di salvare in Local Storage il nome del file che si intende visualizzare in tutto il servizio e quindi questa variabile viene passata in due modi diversi a tutte le funzioni che fanno richieste al server, in modo da specificare non solo l'azione ma su quale risorsa si intende operare.
- Le chiamate client-side alle API vengono gestite in modo asincrono tramite AJAX con jQuery così da non bloccare, né ricaricare l'interfaccia utente.
- Ad ogni modifica dei file, il contenuto verrà rigenerato chiamando le apposite funzioni previa inizializzazione delle tabelle e grafici per evitare conflitti.
- A ogni azione intrapresa dall'utente sui dati si avrà un feedback visivo tramite toast di Bootstrap con colorazione personalizzata.
- Le operazioni potenzialmente dannose richiederanno una conferma prima di essere effettuate.

4.4.4 Service

Questa directory contiene le tre sezioni del servizio, ossia il contenuto html da iniettare nel DOM a seconda della scelta da parte dell'utente.

4.5 Server

Il server si trova al primo livello dalla root, ed è il file che definisce le route per usare il servizio e quindi i metodi CRUD, nonché le azioni che l'utente può intraprendere sui dati persistenti. Questo è il core della business logic del servizio e sfrutta le directory seguenti.

4.5.1 Utils

Questa contiene funzioni per la lettura, scrittura e modifica dei file lato server.

- Al caricamento di ogni file durante il parsing verranno anche controllati i formati di data e ora e normalizzati con le apposite funzioni di utilità lato server.

4.5.2 Upload

In questa directory verranno caricati i file csv e i file json convertiti tramite multer e conterrà anche la lista principale 'files.json' per tenere traccia di tutti i file attualmente presenti in tale directory.

- È stata adottata una gestione che impedisce l'inserimento di file duplicati.
- In seguito al parsing verranno eliminati i sorgenti csv, poiché tutta la manipolazione successiva avverrà solo ed esclusivamente su file json convertiti in oggetti javaScript.
- Si lascia l'attribuzione dell'ente fornitore dei dati a responsabilità dell'utente al momento del caricamento tramite l'apposito form per rispettare l'obbligo imposto dalla licenza (CC BY 4.0).

5 Dati

5.1 Open Data

I dati utilizzati per il testing del web service provengono da:

- **ARPA Marche**:
  `https://www.arpa.marche.it/`

Sono stati tagliati e modificati per scopi dimostrativi, come testing e validazione.

5.2 Licenza

- **Creative Commons - Attribuzione 4.0 Internazionale (CC BY 4.0)**:
  `https://creativecommons.org/licenses/by/4.0/`

Consente la condivisione, copia, redistribuzione, modifica e uso in qualsiasi mezzo o formato per qualsiasi scopo anche commerciale. Con l'obbligo di attribuire un credito adeguato, indicare la licenza e specificare se sono state apportate modifiche ai dati.

6 Documentazione API

Le API del web service sono progettate seguendo il modello Request/Response basato sul protocollo HTTP, aderendo ai principi RESTful. Il servizio è stateless, cioè non mantiene alcuna informazione sullo stato tra richieste consecutive, ogni richiesta è indipendente.

6.1 Endpoints

1. `/files`
   Risponde a una **POST** aggiungendo alla directory upload il file e le informazioni caricate nel body della richiesta.
2. `/files/measurements/{filename}`
   Risponde a una **POST** aggiungendo una nuova misurazione nel file corrente specificato dal path parameter, con i valori passati tramite body della richiesta.
3. `/files/data/{filename}`
   Risponde a una **GET** fornendo il file corrente specificato nel path parameter.
4. `/files/list`
   Risponde a una **GET** fornendo il file contenente la lista dei file caricati.
5. `/files/{filename}`
   Risponde a una **DELETE** eliminado il file con nome passato tramite path parameter.
6. `/files/{date}/{time}/{filename}`
   Risponde a una richiesta **DELETE**, eliminando la misurazione corrispondente all'interno del file indicato dal path parameter **filename**, che contiene una misurazione con **date** e **time** corrispondenti a quelli specificati.
7. `/files/{date}/{time}/{filename}`
   Risponde a una **PUT** modificando una misurazione all'interno del file indicato dal path parameter **filename** con **date** e **time** corrispondenti, aggiornando con i nuovi valori passati nel body della richiesta.

6.2 Formato delle richieste

Il formato delle richieste varia in base al verbo HTTP e alle esigenze specifiche. Per lo scambio dei messaggi, è necessario specificare nell'header della richiesta il campo Content-Type, che indica il tipo MIME del contenuto.

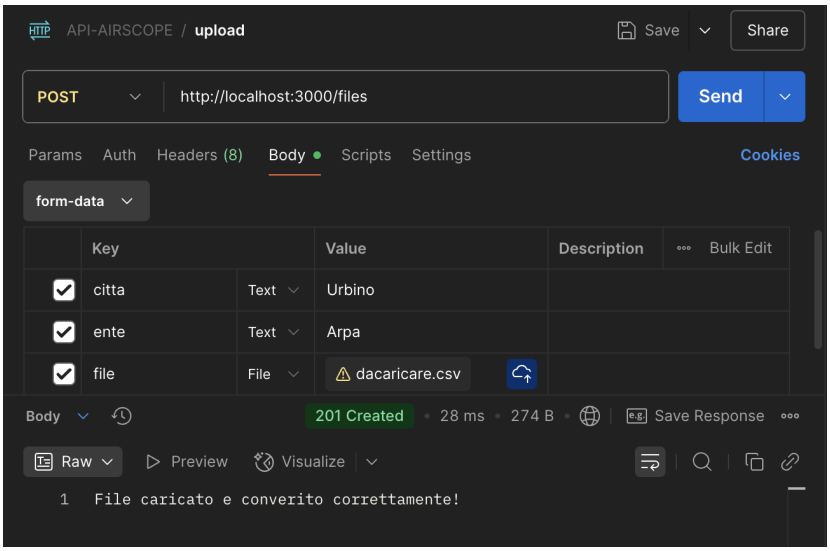
6.2.1 Caricamento file

- **Metodo:** POST
- **URL:** `/files`
- **Content-Type:** `multipart/form-data`
- **Struttura della richiesta:**

```
curl --location 'http://localhost:3000/files' \
--form 'citta="Urbino"' \
--form 'ente="Arpa"' \
--form 'file=@"/Users/macos/Desktop/L-31/TWEB/airscope/public
   /assets/dacaricare.csv"'
```

- **Risposta: 201 Created** `'text/plain'`

```
File caricato e converito correttamente!
```



Il caricamento del file avviene tramite il verbo POST ed utilizza il middleware multer per fornire al server il body, la risposta viene inviata come testo per essere riutilizzata come feedback verso l'utente con i bootstrap toast.

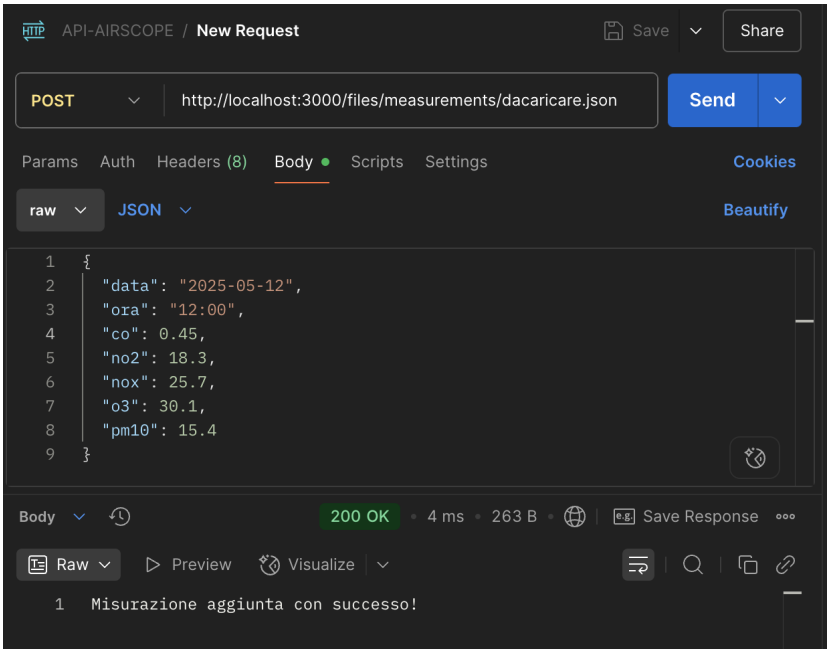
6.2.2 Aggiunta misurazione a file esistente

- **Metodo:** POST
- **URL:** `/files/measurements/{filename}`
- **Content-Type:** `application/json`
- **Struttura della richiesta:**

```
{
  "data": "2025-05-12",
  "ora": "12:00",
  "co": 0.45,
  "no2": 18.3,
  "nox": 25.7,
  "o3": 30.1,
  "pm10": 15.4
}
```

- **Risposta: 200 OK** `'text/plain'`

```
Misurazione aggiunta con successo!
```

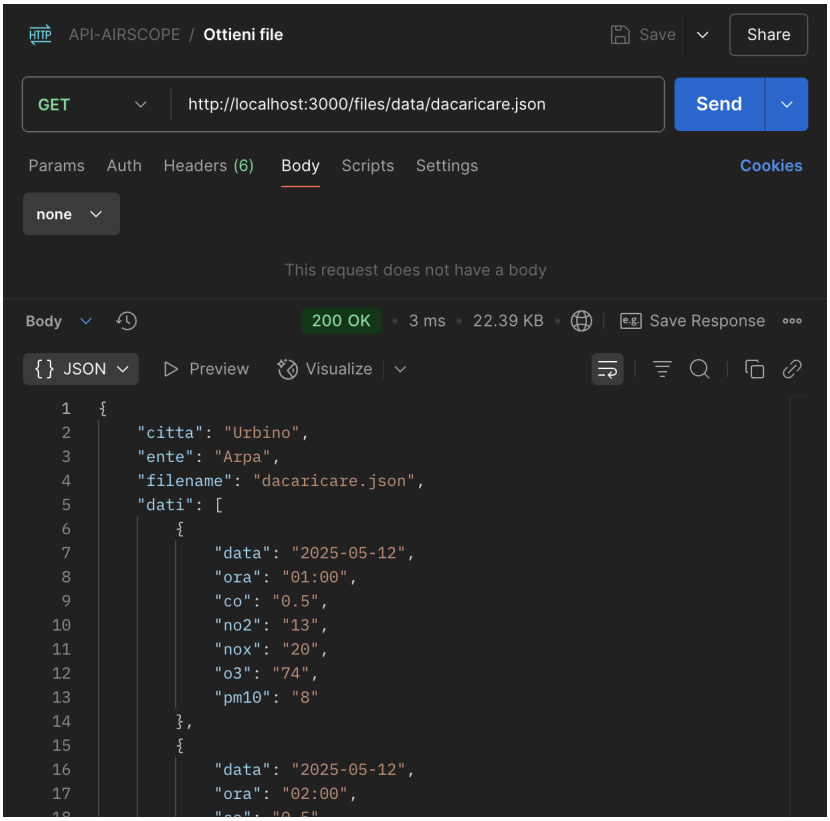


L'aggiunta della misurazione si effettua tramite il verbo POST, e si passa come path parameter il nome del file al quale si vuole aggiungere la misurazione, la risposta è sempre testuale per essere riutilizzata come feedback da fornire all'utente.

6.2.3 Ottenimento file specifico

- **Metodo:** GET
- **URL:** `/files/data/{filename}`
- **Risposta: 200 OK** `'application/json'`

```
{
"citta": "Urbino",
"ente": "Arpa",
"filename": "dacaricare.json",
"dati": [
    {
        "data": "2025-05-12",
        "ora": "01:00",
        "co": "0.5",
        "no2": "13",
        "nox": "20",
        "o3": "74",
        "pm10": "8"
    }, ...
```

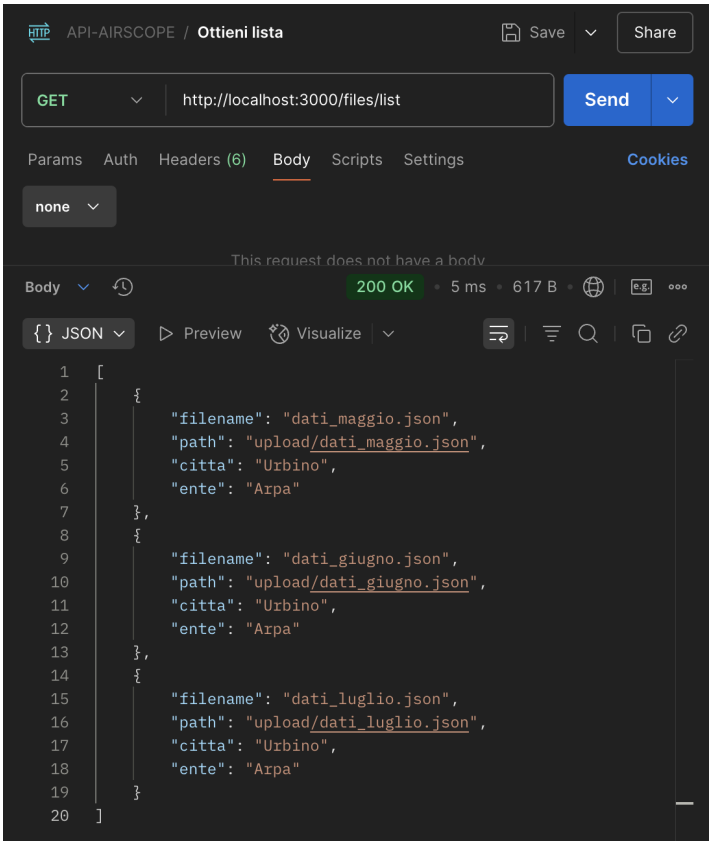


La richiesta per ottenere il file utilizza il metodo GET e come path parameter il nome del file desiderato, restituisce in formato json il contenuto del file al client.

6.2.4 Ottenimento lista file

- **Metodo:** GET
- **URL:** `/files/list`
- **Risposta: 200 OK** `'application/json'`

```
[
{
    "filename": "dati_maggio.json",
    "path": "upload/dati_maggio.json",
    "citta": "Urbino",
    "ente": "Arpa"
},
{
    "filename": "dati_giugno.json",
    "path": "upload/dati_giugno.json",
    "citta": "Urbino",
    "ente": "Arpa"
}, ...
]
```

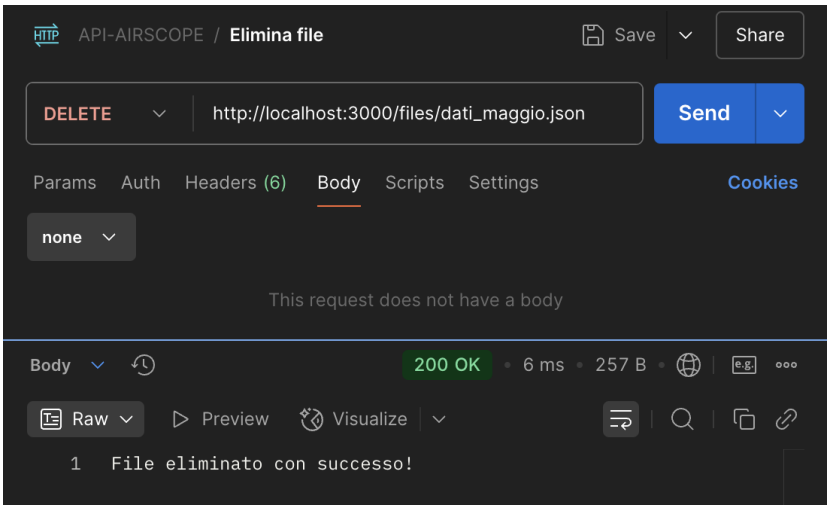


La risposta contiene il contenuto del file in formato json dove viene tenuta traccia di tutti i file attualmente disponibili nella directory upload.

6.2.5 Eliminazione file

- **Metodo:** DELETE
- **URL:** `/files/{filename}`
- **Risposta: 200 OK** `'text/plain'`

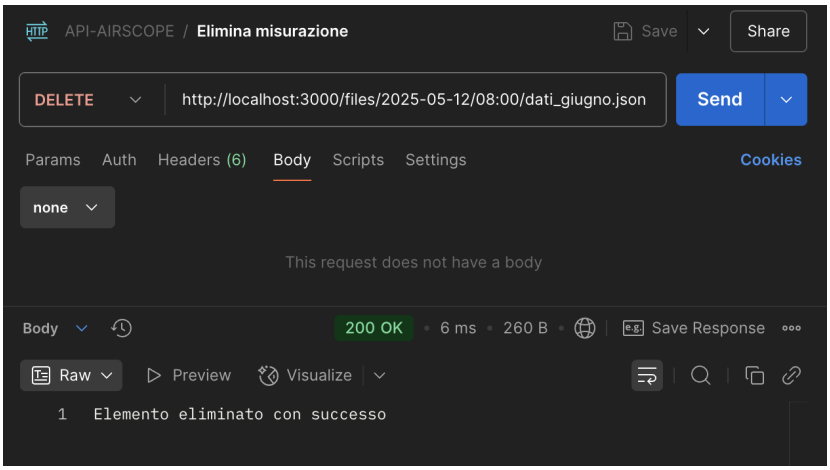
```
File eliminato con successo!
```



6.2.6 Eliminazione misurazione specifica

- **Metodo:** DELETE
- **URL:** `/files/{date}/{time}/{filename}`
- **Risposta: 200 OK** `'text/plain'`

```
Misurazione eliminata con successo
```



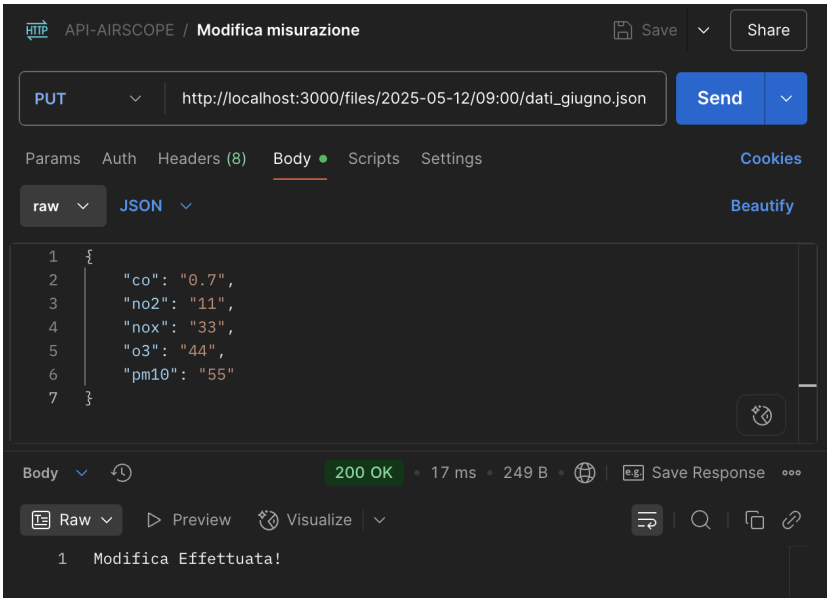
6.2.7 Modifica misurazione specifica

- **Metodo:** PUT
- **URL:** `/files/{date}/{time}/{filename}`
- **Content-Type:** `'application/json'`
- **Struttura della richiesta:**

```
{
    "co": "0.7",
    "no2": "11",
    "nox": "33",
    "o3": "44",
    "pm10": "55"
}
```

- **Risposta: 200 OK** `'text/plain'`

```
Modifica Effettuata!
```

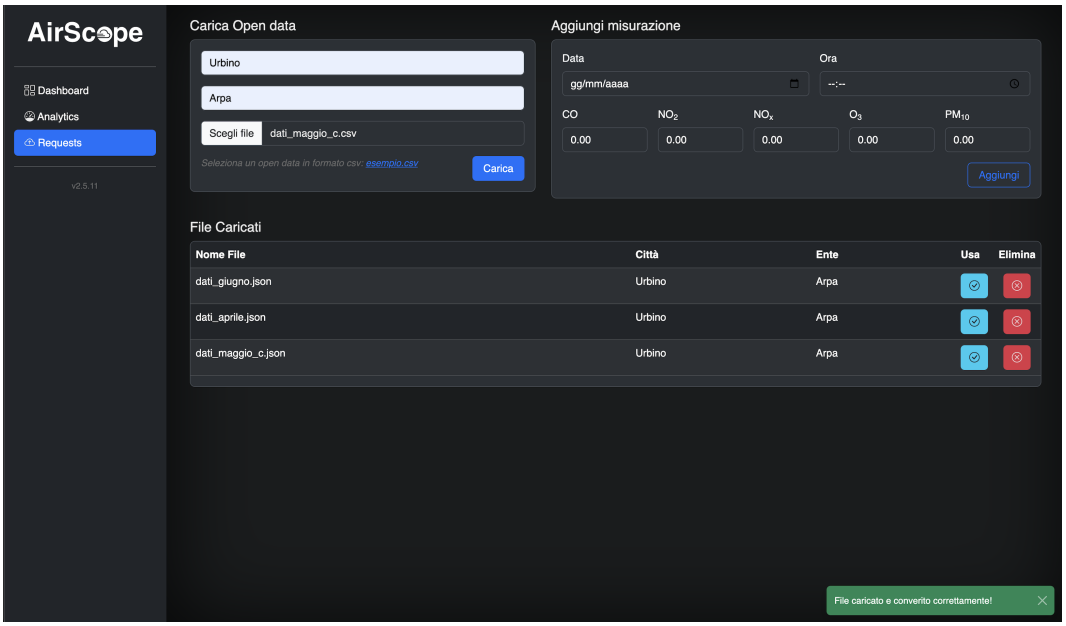


Il metodo PUT viene utilizzato per effettuare le modifiche delle misurazioni su uno specifico file e la misurazione da aggiornare è quella che corrisponde in data ed ora ai path parameter passati nell'URL.

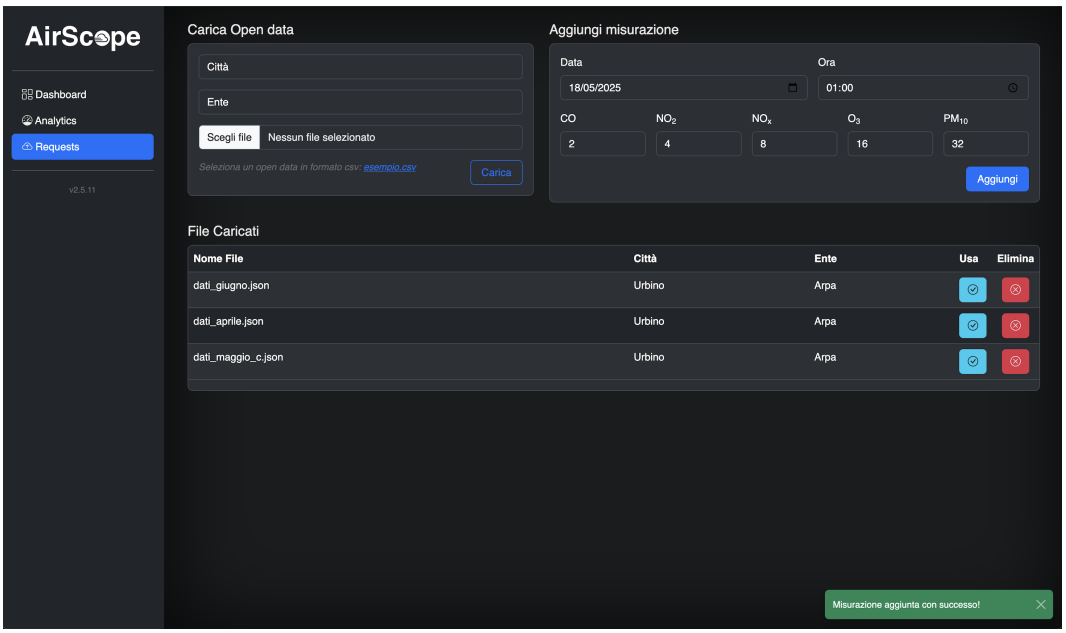
7 Testing End-to-End

Il testing End-to-End (E2E) viene effettuato nella fase finale per verificare l'intero flusso dell'applicazione, simulando le interazioni tipiche dell'utente finale. In questa fase si testano gli scenari già presentati, ma eseguiti direttamente tramite l'interfaccia del browser, garantendo così il corretto funzionamento di tutti i componenti.

1. **Caricamento file**

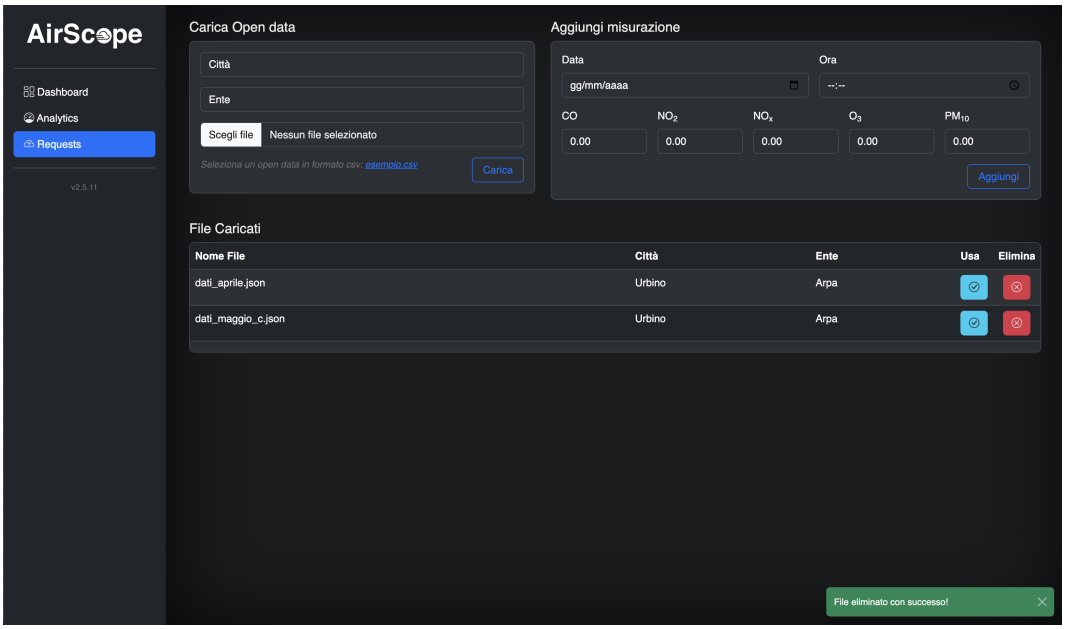


2. **Aggiunta misurazione a file**

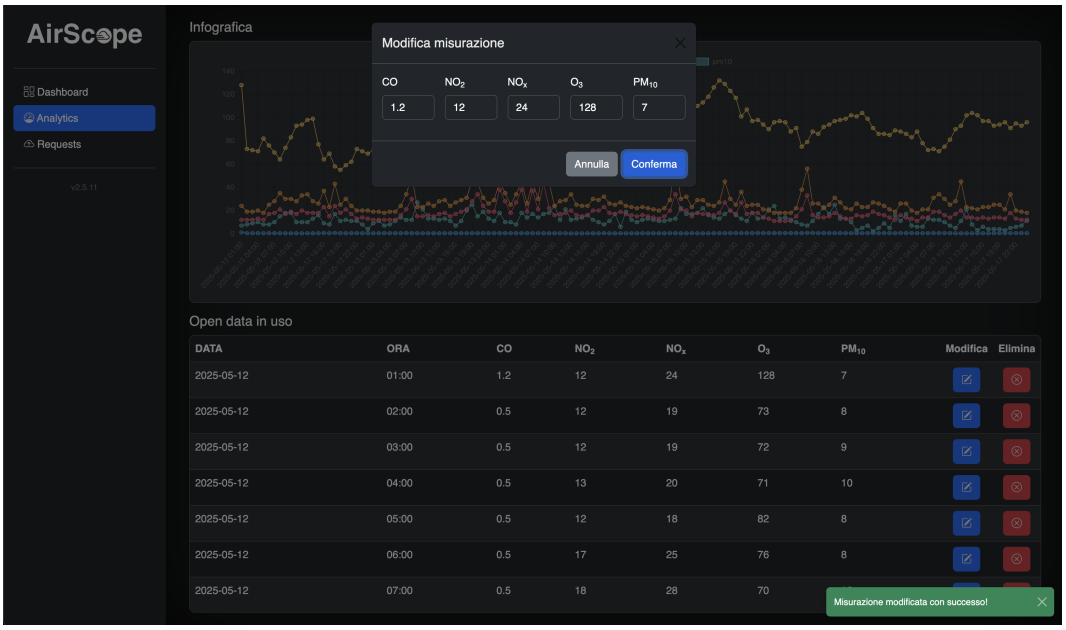


3. **Ottenimento file con misurazioni ed eliminazione misurazione**

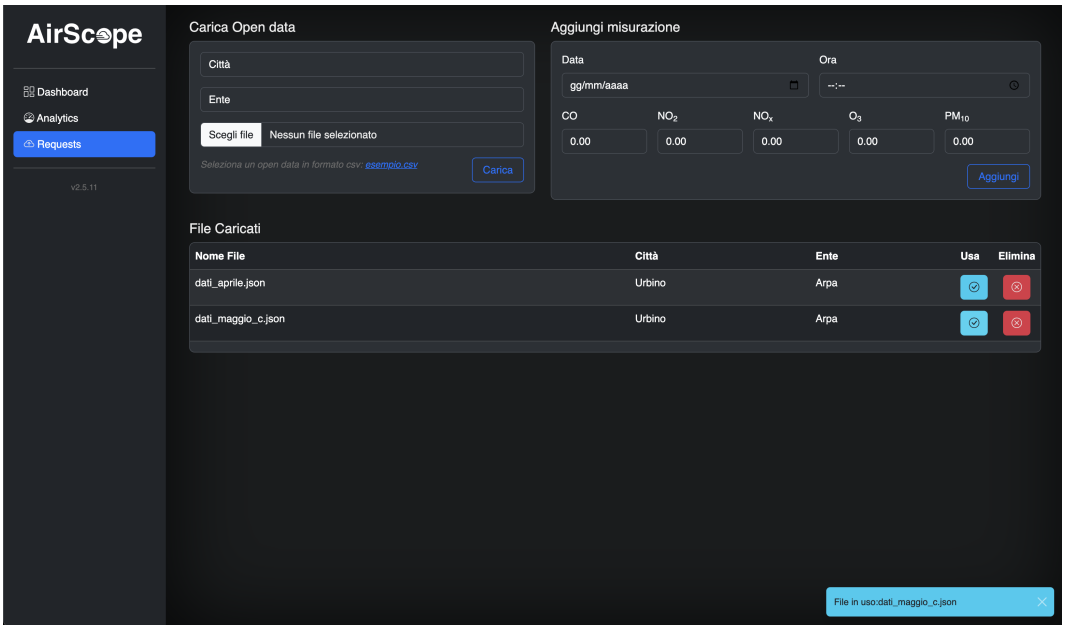
4. **Ottenimento lista file ed eliminazione file**



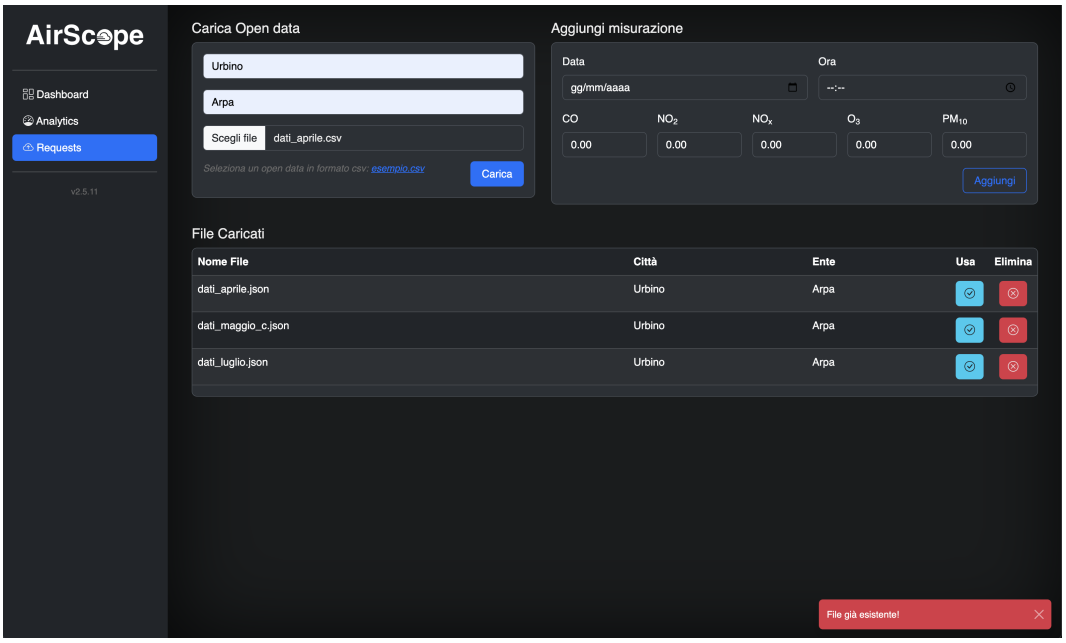
5. Modifica misurazione



6. Cambio file in utilizzo



7. Gestione dei duplicati



8. Errore di parsing