TrafficFlowManager

Esame di Big Data Architectures (9 CFU)

Indice

1 Introduzione

Questa relazione descrive il progetto TrafficFlowManager, portato avanti nella prima metà del 2021 per l'esame di Big Data Architectures (9 CFU). Il contesto è quello della gestione e visualizzazione di ricostruzioni di traffico realtime su mappe interattive. Gli obiettivi erano:

- Migliorare la parte di visualizzazione sulla mappa, aumentando la densità dei segmenti in modo da poter scalare da una risoluzione all'altra senza avere problemi di discontinuità (evitando quell'effetto di linee *disgiunte* in seguito allo zoom, dovuto all'eccessiva approssimazione);

- Realizzare una parte di gestione delle informazioni (*scenari*) dinamica, che permettesse in maniera immediata sia il caricamento di nuovi flussi di traffico che lo scorrimento dei dati delle singole ricostruzioni e la possibilità di muoversi tra scenari diversi.

La soluzione proposta è stata sviluppata in locale su macchine virtuali **Snap4city Alone/MAIN versione 1.5** [1] e **GISGeoServer** [2], ed è poi stata integrata in produzione all'interno della piattaforma Snap4City.

Il progetto è stato sviluppato con MacBook Pro (Retina, 13-inch, Mid 2014) con processore 2,6 GHz Dual-Core Intel Core i5 e 8 GB di RAM (1600 MHz DDR3), con sistema operativo macOS Big Sur (versione 11.3.1). Gli IDE e i software utilizzati sono:

- *IntelliJ IDEA 2021.1.1 (Ultimate Edition)* per Java
- *PhpStorm 2021.1.2* per HTML, PHP e JavaScript
- GeoServer 2.14.2
- Apache Tomcat/8.0.32
- Java 1.8.0_191

2 Backend

2.1 Estrazione di Shapefile da flusso JSON

I dati che vengono inviati all'applicazione in input sono principalmente di due tipi:

- **Grafo statico**: fornisce informazioni circa la struttura del grafo stradale sotteso al modello di ricostruzione del traffico, ed è fisso per ogni città/scenario;
- **Ricostruzione temporale**: contiene le informazioni circa i valori di densità realtime relative alla ricostruzione del traffico associata ad un certo grafo statico, avvenuta in un determinato istante temporale. Sono presenti anche dei metadati che descrivono la ricostruzione, quali il nome del grafo statico associato, il luogo, l'ora, il tipo di scenario e l'unità di misura.

I grafi statici vengono inviati al server una volta sola, e vengono riutilizzati; le ricostruzioni sono invece periodiche (ad es. ogni 15 minuti) perchè rappresentano il traffico ad un certo istante temporale.

Ogni ricostruzione, fornita al server in formato JSON, deve quindi essere convertita in un formato adatto sia alla memorizzazione che alla visualizzazione su mappa in maniera efficiente. La soluzione sviluppata si appoggia a GeoServer [3], un server open source che permette di condividere ed elaborare dati geospaziali. Essendo dati di tipo vettoriale, il formato ideale per le ricostruzioni è lo Shapefile, in quanto permette di descrivere spazialmente segmenti e polilinee, ed è compatibile nativamente con GeoServer.

L'applicazione è stata sviluppata in linguaggio Java utilizzando *Maven* come package manager e *Javax JSON* [4] come libreria per il parsing del JSON. Come database è stato utilizzato un file JSON, gestito da classi Java apposite tramite la stessa libreria *Javax*.

2.1.1 Parametri di configurazione

L'applicazione necessita di alcuni parametri esterni, che vengono definiti in un file *config.properties* che dovrà essere caricato manualmente sul server. Di seguito un esempio di configurazione iniziale:

```
geoServerUrl=http://localhost:8080/geoserver/rest
geoServerUser=admin
geoServerPass=geoserver
geoServerWorkspace=traffic
geoServerStyleName=road_traffic_style
staticGraphsFolder=/home/debian/tfm/static_graphs
tmpLayersFolder=/home/debian/tfm/tmp_layers
reconstructionsFolder=/home/debian/tfm/reconstructions
db=/home/debian/tfm/reconstructions.json
```

Estratto 1: Esempio di file di configurazione

I parametri sono i seguenti:

- **geoServerUrl**: è l'url per le API REST del GeoServer (volendo potrebbe essere installato anche su un'altra macchina);
- **geoServerUser** e **geoServerPass**: credenziali di accesso al GeoServer, utilizzate per il caricamento dei layer;
- **geoServerWorkspace**: workspace del GeoServer in cui verranno caricati i layer (deve essere creato manualmente);
- **geoServerStyleName**: nome dello stile presente su GeoServer con cui colorare i layer; nella soluzione proposta si tratta di uno stile "*attribute-based*" [5] che assegna un colore al segmento in base ad un'etichetta, a sua volta ottenuta in base al valore di densità di traffico e al numero di corsie;
- **staticGraphsFolder**: path della cartella in cui vengono salvati i grafi statici;
- **tmpLayersFolder**: path della cartella in cui vengono elaborati temporaneamente gli Shapefile, prima di essere caricati;
- **reconstructionsFolder**: path della cartella in cui vengono salvati i JSON compressi delle ricostruzioni;
- **db**: path del file JSON utilizzato come database.

2.1.2 Estrazione

L'operazione di estrazione e caricamento dello Shapefile avviene seguendo questi passi:

- Il JSON in input viene unito con il grafo statico in modo da estrarre un singolo file CSV, in cui ogni riga corrisponde ad un segmento, contenente anche tutti i suoi metadati (id, coordinate di inizio/fine, valore di densità di traffico);
- Utilizzando la libreria **GeoTools** [6], il file CSV viene iterato, e ciascun segmento (righe della tabella) viene convertito in formato geospaziale (Shapefile) tramite le sue coordinate;

- Il passo precedente genera un insieme di file (con estensioni .dbf, .fix, .prj, .shp e .shx) che rappresentano geograficamente la ricostruzione. Questi file vengono quindi compressi in un unico .zip che viene caricato su GeoServer tramite le API apposite;
- Terminato il caricamento, vengono quindi cancellati tutti i file intermedi (il CSV, i vari file dello Shapefile e l'archivio finale).

Prima del caricamento su GeoServer, viene assegnato a ciascuna ricostruzione un identificativo univoco (*layerName*) costituito dalla tripletta luogo - scenario - ora.

2.1.3 Salvataggio delle ricostruzioni JSON originali

Un ulteriore requisito prevedeva che ciascuna ricostruzione JSON originale dovesse essere sempre reperibile, specialmente in ottica futura di comparazione degli scenari di traffico. Per questo tutte le ricostruzioni JSON vengono anche salvate, univocamente, in una cartella (*reconstructionsFolder*) e potranno essere richieste tramite API REST (vedi 2.2.3). Inizialmente l'idea era quella di salvare tutti i file originali, ma la quantità di memoria necessaria non era trascurabile (alcune ricostruzioni possono arrivare a pesare anche 17MB - 2.5GB al giorno generandone una ogni 10 minuti). Per questo prima di essere salvate vengono compresse in formato zip, che garantisce un risparmio di memoria del 3700% (da 17MB a 450KB).

2.2 API REST

Le API sono state realizzate con Java Servlet e compilate in un unico file (*trafficflowmanager.war*) di cui fare il deploy su *Apache Tomcat*. Di seguono vengono descritti i tre endpoint messi a disposizione tramite servizio REST e il loro funzionamento.

2.2.1 API Rest per il caricamento dei flussi

Le funzioni di caricamento di grafi statici e ricostruzioni sono fornite tramite servizio REST ed accessibili con API apposite su endpoint **/api/upload**. In particolare:

- **Caricamento di un grafo statico**
  - **Endpoint**: /api/upload?type=staticGraph
  - **Metodo**: POST
  - **Content-Type**: application/json
  - **Corpo**: grafo statico in formato JSON, ad esempio:

```
{
  "nameGraphID": {
    "staticGraphName": "
       FirenzeTrafficRealtimeStaticGraph"
  },
  "dataGraph": [
    {
      "road": "OS00024605529SR",
      "segments": [
        {
```

```
            "id": "OS00529RE/5--OS00024E/6.1",
            "start": {
              "long": "11.283867835999",
              "lat": "43.787147521973"
            },
            "end": {
              "long": "11.283638191223",
              "lat": "43.7870773315434"
            }
          }, ...
        ]
      }, ...
  ]
}
```

Estratto 2: Esempio di grafo statico in JSON

- **Risposta (JSON)**:
  - {"success": true} e codice 200, oppure
  - {"success": false, "error": "descrizione errore"} e codice 400

Questa chiamata non fa nessun processamento, ma salva solamente il file JSON in una cartella predefinita in modo da riutilizzarlo in futuro.

- **Caricamento di una ricostruzione del traffico**

  Un prerequisito di questa chiamata è la presenza del grafo statico corrispondente, che deve essere già stato caricato in precedenza.

  - **Endpoint**: /api/upload?type=reconstruction
  - **Metodo**: POST
  - **Content-Type**: application/json
  - **Body**: ricostruzione in formato JSON, ad esempio:

```
{
  "metadata": {
    "fluxName": "FirenzeTrafficRealtime",
    "locality": "FirenzeFIPILI",
    "organization": "Toscana",
    "scenarioID": "TrafficRealtime",
    "dateTime": "2021-04-27T16:41:00",
    "duration": "10min",
    "metricName": "TrafficDensity",
    "unitOfMeasure": "vehicle per 20m",
    "colorMap": "densityTrafficMap",
    "staticGraphName": "
       FirenzeTrafficRealtimeStaticGraph"
  },
  "reconstructionData": {
    "OS00024605529SR": {
      "data": [
```

```
            {
              "OS00529RE/5--OS00024RE/6.1": "0.312",
              ...
            }
        ]
    }, ...
  }
}
```

Estratto 3: Esempio di ricostruzione del traffico in JSON

- **Risposta (JSON)**:
  - {"success": true} e codice 200, oppure
  - {"success": false, "error": "descrizione errore"} e codice 400

Questa chiamata avvia il procedimento descritto alla sezione precedente per la conversione in Shapefile e il caricamento del layer sul GeoServer.

2.2.2 API Rest per i metadati

Un altro endpoint, **/api/metadata**, è stato riservato alle operazioni sui metadati, sia in lettura che scrittura.

- **Metadati di tutti i flussi presenti nel database**
  - **Endpoint**: /api/metadata
  - **Metodo**: GET
  - **Risposta**: elenco JSON di tutti i flussi presenti nel database, ad esempio:

```
[
  {
    "fluxName": "FirenzeTrafficRealtime",
    "locality": "FirenzeFIPILI",
    "organization": "DISIT",
    "scenarioID": "TrafficRealtime",
    "colorMap": "densityTrafficMap",
    "instances": 3,
    "metricName": "TrafficDensity",
    "unitOfMeasure": "vehicle per 20m",
    "staticGraphName": "FirenzeTrafficStaticGraph"
  },
  ...
]
```

Estratto 4: Esempio di risposta

- **Metadati dei layer di un flusso specifico**
  - **Endpoint**: /api/metadata?fluxName={NOME FLUSSO}
  - **Metodo**: GET

- **Risposta**: elenco JSON di tutti i layer del flusso specificato presenti nel database, ad esempio:

```
[
  {
    "fluxName": "FirenzeTrafficRealtime",
    "locality": "FirenzeFIPILI",
    "organization": "DISIT",
    "scenarioID": "TrafficRealtime",
    "dateTime": "2021-04-01T16:25:00",
    "duration": "10min",
    "metricName": "TrafficDensity",
    "unitOfMeasure": "vehicle per 20m",
    "colorMap": "densityTrafficMap",
    "staticGraphName": "FirenzeTrafficStaticGraph",
    "layerName": "FirenzeFIPILI_TrafficRealtime_2021
       -04-01T16-25-00"
  },
  ...
]
```

Estratto 5: Esempio di risposta

- **Eliminare un singolo layer dal database**

  Questo metodo consente di eliminare un singolo layer (es: FirenzeFIPILI_TrafficRealtime_2021-04-01T16-25-00) dal database.

  - **Endpoint**: /api/metadata?action=delete_data&id={NOME LAYER}
  - **Metodo**: POST
  - **Risposta**: 200 OK

- **Eliminare un intero flusso dal database**

  Questo metodo consente di eliminare un intero flusso (es: FirenzeTrafficRealtime) dal database.

  - **Endpoint**: /api/metadata?action=delete_metadata&id={NOME FLUSSO}
  - **Metodo**: POST
  - **Risposta**: 200 OK

- **Cambiare la colorMap di un flusso**

  Questo metodo consente invece di cambiare il campo *colorMap* di ciascun layer del flusso specificato.

  - **Endpoint**: /api/metadata?action=change_color_map&id={FLUSSO}&valore={VALORE}
  - **Metodo**: POST
  - **Risposta**: 200 OK

2.2.3 API Rest per i flussi JSON

Infine l'ultimo endpoint **/api/json** è stato aggiunto per il recupero di flussi e grafi statici originali in formato JSON.

- **Ottenere il JSON originale di un grafo statico**
  - **Endpoint**: /api/json?staticGraphName={NOME GRAFO STATICO}
  - **Metodo**: GET
  - **Risposta**: il JSON originale del grafo statico (errore 400 se non esiste nel database).
- **Ottenere il JSON originale di un grafo statico**

  Essendo le ricostruzioni salvate in formato compresso (zip), questa chiamata provvederà a decomprimere il file al volo prima di servire il contenuto in formato JSON.
  - **Endpoint**: /api/json?layerName={NOME LAYER}
  - **Metodo**: GET
  - **Risposta**: il JSON originale della ricostruzione (errore 400 se non esiste nel database).

3 Front-End

Lo sviluppo della parte di front-end è stato suddiviso in due parti:

- Creazione della tabella per la gestione dei flussi all'interno della dashboard Snap4City, tramite modulo *process-loader* [7];
- Gestione della visualizzazione e controllo dei flussi su mappa interattiva all'interno della dashboard Snap4City, tramite modulo *dashboard-frontend* [8].

3.1 Tabella per la gestione dei flussi

La tabella è stata sviluppata e introdotta nella dashboard Snap4City aggiungendo la voce *TrafficFlowManager* al menu *Resource Manager* (parte gestita dal modulo *process-loader*). È stato prodotto quindi un singolo file php (*trafficflowmanager.php*) da caricare al percorso */process-loader/www/*; la sezione è stata aggiunta al menu modificando direttamente il database SQL, aggiungendo una riga alle tabelle *MainMenuSubmenus*/*MobMainMenuSubMenus* in modo da reindirizzare al file php.

L'aspetto e il funzionamento sono stati pensati in maniera del tutto analoga alla preesistente tabella *HeatMap Manager* (vedi figura 1). Le caratteristiche sono:

- Visualizzazione dell'elenco dei flussi presenti nel sistema, con paginazione;
- Visualizzazione dei dettagli di ciascun flusso (nome, luogo, organizzazione, scenario, istanze, metrica, unità di misura, nome del grafo statico associato);
- Possibilità di visionare le singole ricostruzioni di ogni flusso, con paginazione, tramite pulsante VIEW DATA (vedi figura 2) che riporta data, nome, durata;
- Possibilità di visualizzare il JSON di ciascuna ricostruzione tramite pulsante VIEW JSON (vedi figura 2);

- Possibilità di eliminare sia una singola ricostruzione che un intero flusso dal database, tramite pulsante DEL;
- Possibilità di visualizzare e cambiare la color map di ciascun flusso.

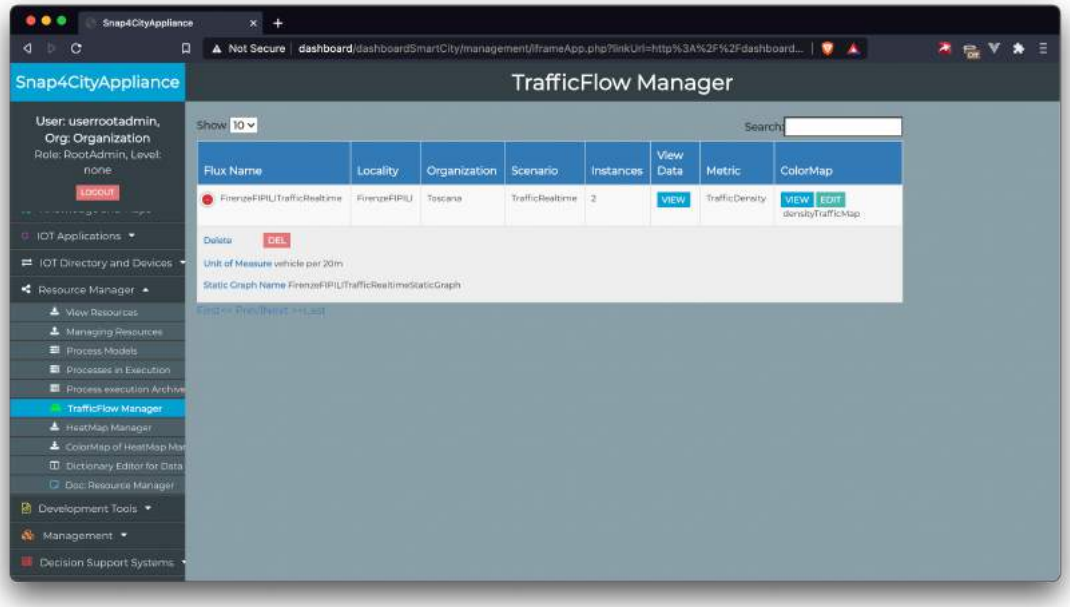

Figura 1: Tabella per la gestione dei flussi

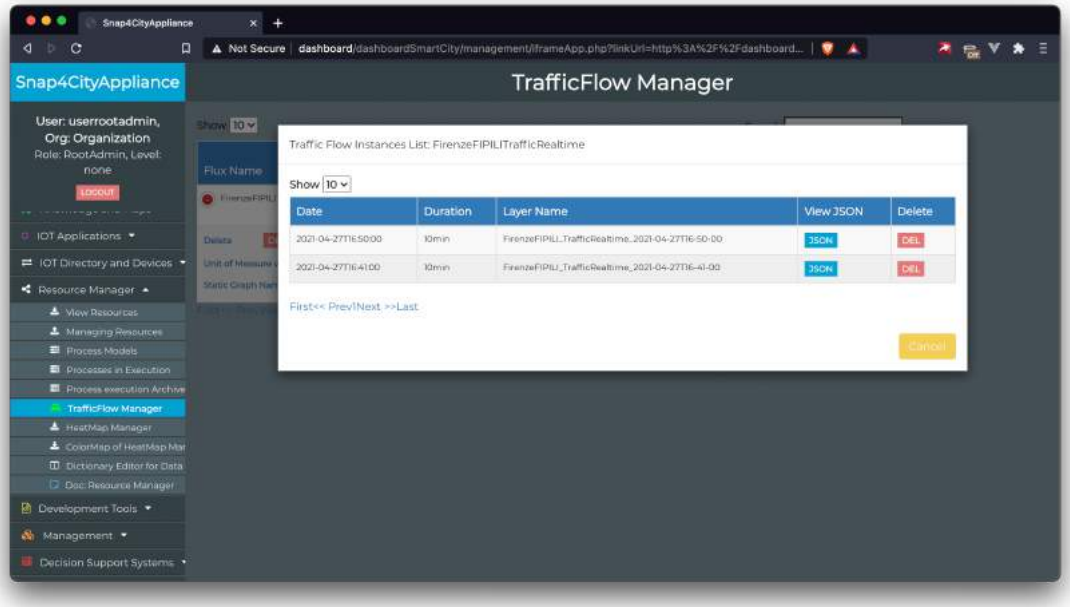

Figura 2: Visualizzazione delle singole ricostruzioni di un flusso

3.2 Visualizzazione dei flussi su mappa interattiva

Per aggiungere il supporto alla visualizzazione di queste nuove heatmap di traffico sono stati modificati i file *widgetMap.php* e *widgetSelectorNew.php* in

/dashboard-builder/dashboard_frontend/widgets/.

Nuovi scenari di traffico possono essere aggiunti ad un widget tramite il menu "*More options...*" dal selettore della dashboard, inserendo una nuova entry con un URL del tipo:

```
1 http://geoserver:8080/geoserver/traffic/wms?service=WMS
2   &layers=FirenzeFIPILITrafficRealtime&trafficflowmanager=true
```

Estratto 6: Esempio di URL per aggiunta scenario di traffico alla dashboard

Il parametro specificato dalla chiave *layers* (*FirenzeFIPILITrafficRealtime* nell'esempio) specifica il nome del flusso da mostrare; il parametro *trafficflowmanager=true* invece serve a distinguere heatmap di traffico da heatmap tradizionali.

Le funzionalità del widget sono:

- Visualizzazione dei flussi in realtime (figura 3) in maniera efficiente tramite protocollo WMS su GeoServer, con colorazione dei segmenti in base alla densità di traffico (verde, giallo, arancione, rosso);
- Possibilità di cambiare l'opacità del layer;
- Possibilità di scorrere le ricostruzioni del traffico precedenti a quella corrente (pulsante *Prev*);
- Possibilità di visualizzare l'andamento del traffico della giornata tramite una animazione di 24 ore (figura 4). Per regolare la risoluzione dell'animazione, è possibile aggiungere il parametro *width* all'URL dello scenario, specificando la larghezza desiderata in pixel (es: *width=1024*);
- Possibilità di sovrapporre altre heatmap a quelle di traffico, in maniera additiva. Questo può risultare utile per comparare diversi scenari contemporaneamente: le due heatmap sono indipendenti l'una dall'altra e ognuna ha la propria finestra di controllo (figura 5).

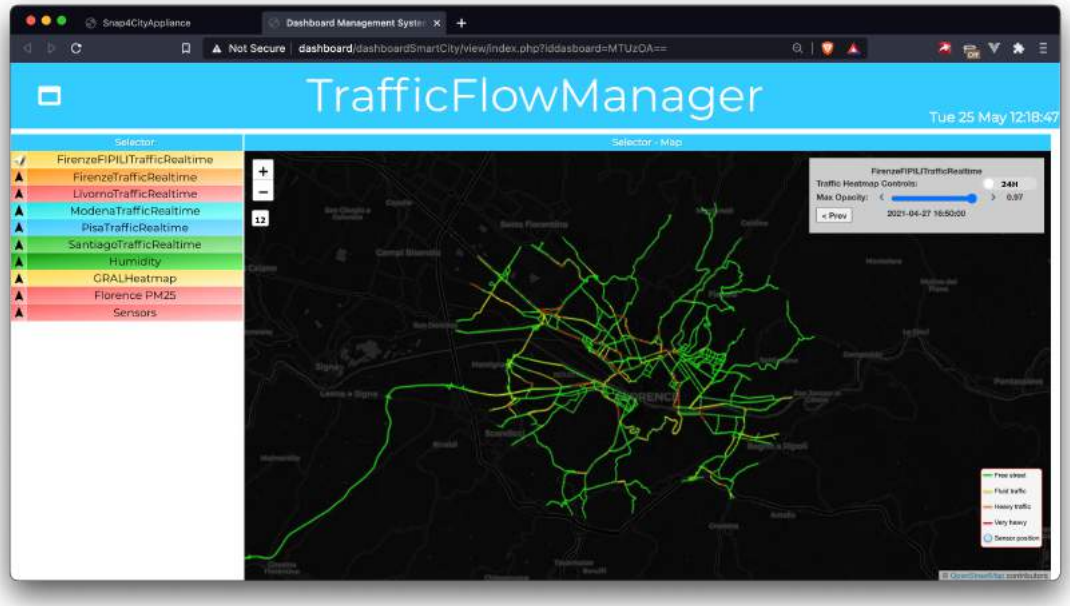

Figura 3: Visualizzazione dei flussi su mappa

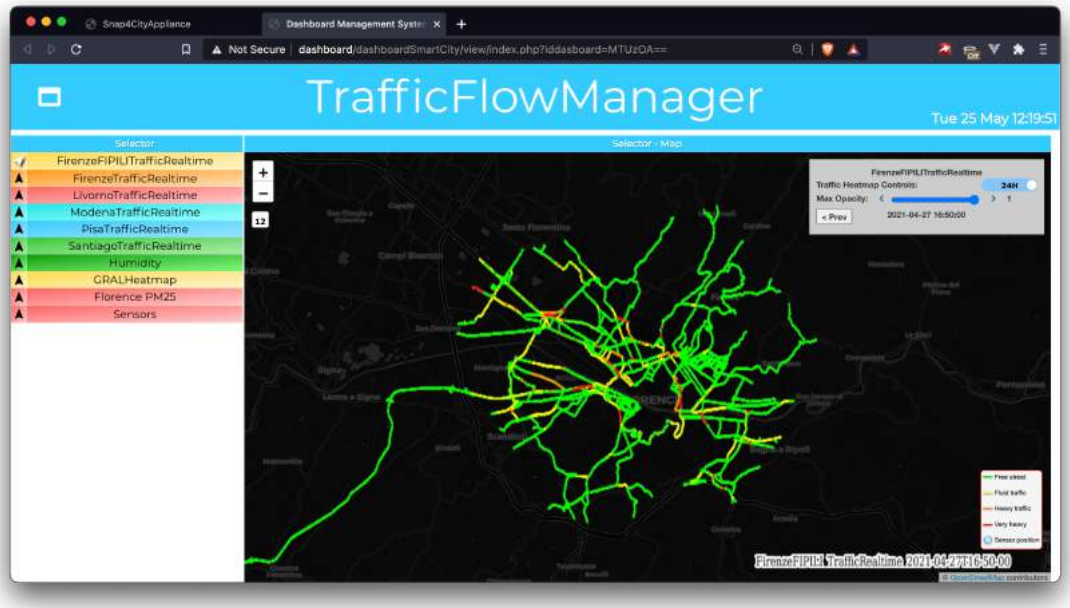

Figura 4: Animazione del flusso a 24 ore

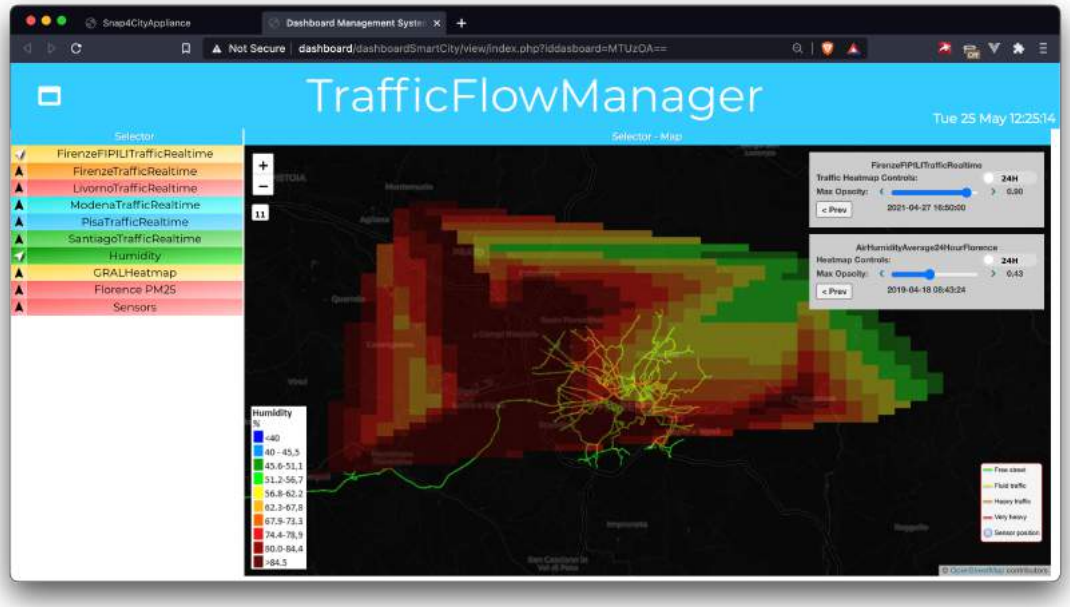

Figura 5: Esempio di sovrapposizione flusso traffico e heatmap umidità

Riferimenti bibliografici

[1] Snap4CityMAIN. https://www.snap4city.org/drupal/node/471.

[2] GISGeoServer VM. https://www.snap4city.org/drupal/node/536.

[3] GeoServer. http://geoserver.org.

[4] Javax JSON. https://mvnrepository.com/artifact/javax.json/javax.json-api.

[5] GeoServer - Attribute-based line style. https://docs.geoserver.org/latest/en/user/styling/sld/cookbook/lines.html#attribute-based-line.

[6] GeoTools. https://geotools.org.

[7] process-loader. https://github.com/disit/process-loader.

[8] dashboard-frontend. https://github.com/disit/dashboard-builder/tree/master/dashboard_frontend.