Interreg

UNION EUROPÉENNE
UNIONE EUROPEA

MOBI MART

MARITTIMO–IT FR–MARITIME

Fonds européen de développement régional
Fondo Europeo di Sviluppo Regionale

PROGETTO MOBIMART
ID prodotto T3.2.2
Modulo specifico della piattaforma

Marzo 2021 - Pisa

REGIONE TOSCANA
REGIONE LIGURIA
REGIONE AUTÒNOMA DE SARDIGNA
REGIONE AUTONOMA DELLA SARDEGNA
RÉGION SUD PROVENCE ALPES CÔTE D'AZUR
CORSE CORSICA
COMUNE DI GENOVA
PROVINCIA DI LIVORNO
Autorità di Sistema Portuale del Mare di Sardegna
Autorità di Sistema Portuale del Mar Tirreno Settentrionale

La cooperazione al cuore del Mediterraneo
La coopération au cœur de la Méditerranée

Progetto numero	168
Acronimo	MOBIMART
Titolo progetto	Mobilità intelligente mare terra
Inizio/ fine progetto	01.04.2018- 30.06.2021 + 120 gg
Durata	39 + 120 gg

Componente	T3 - Sistemi informativi provinciali e comunali
Attività a cui si riferisce il prodotto	T3.2 Sistema informativo dei trasporti e della mobilità
Titolo prodotto	Modulo specifico della piattaforma
Stage di riferimento	P6
Periodo riferimento (inizio/Fine)	01/10/2020 – 31/03/2021

Autore	**AEDIT s.r.l.** Via Fornace Braccini 8 – 56025 Pontedera (PI) P.IVA: 01814780464 Pisamo Srl Comune di Pisa
Versione	1.1
Data	23/03/2021
Responsabile della validazione	Comune di Pisa
Data revisione	
Riepilogo modifiche	
Autore	

modifiche	

Indice/Index

1. Introduzione

Il presente documento contiene la descrizione delle attività svolte nel progetto a partire dal 1 Agosto 2020 al 31 Marzo 2021. Dopo aver preso familiarità con la piattaforma Snap4city, abbiamo dapprima sviluppato alcune procedure per il data ingestion e per la creazione delle Dashboard per la visualizzazione dei dati precedentemente caricati. E' stata creata una Virtual Machine (VM) sul nostro server dove è stata installata l'ultima versione disponibile della piattaforma. Abbiamo progressivamente aggiunto sorgenti di dati che sono stati regolarizzati attraverso l'esecuzione di procedure temporizzate.

Al tempo stesso abbiamo iniziato le attività di visualizzazione dei dati attraverso lo sviluppo di dashboard specifiche per tipologie di dati.

Per tutta la durata del periodo siamo stati in contatto con il supporto dell'Università di Firenze che ha svolto attività di formazione all'inizio del progetto e di consulenza sull'uso della piattaforma durante lo stesso.

2. B.1 Realizzazione del Sito Web delle Azioni di PISAMO/PISA per MobiMart

Il contratto prevede la creazione di un sito web con la descrizione del progetto MobiMart e dove poter pubblicare le varie deliverables del progetto.

Per questo scopo, è stato scelto di utilizzare il content manager Drupal, una piattaforma software opensource che permette la creazione e la distribuzione di complessi siti web dinamici. Drupal è stato scelto per la grande possibilità di personalizzazione grazie alla sua struttura basata su moduli e temi. Viene inoltre garantita la sicurezza grazie alla folta comunità Drupal e su rilasci di numerosi aggiornamenti e risoluzione bugs.

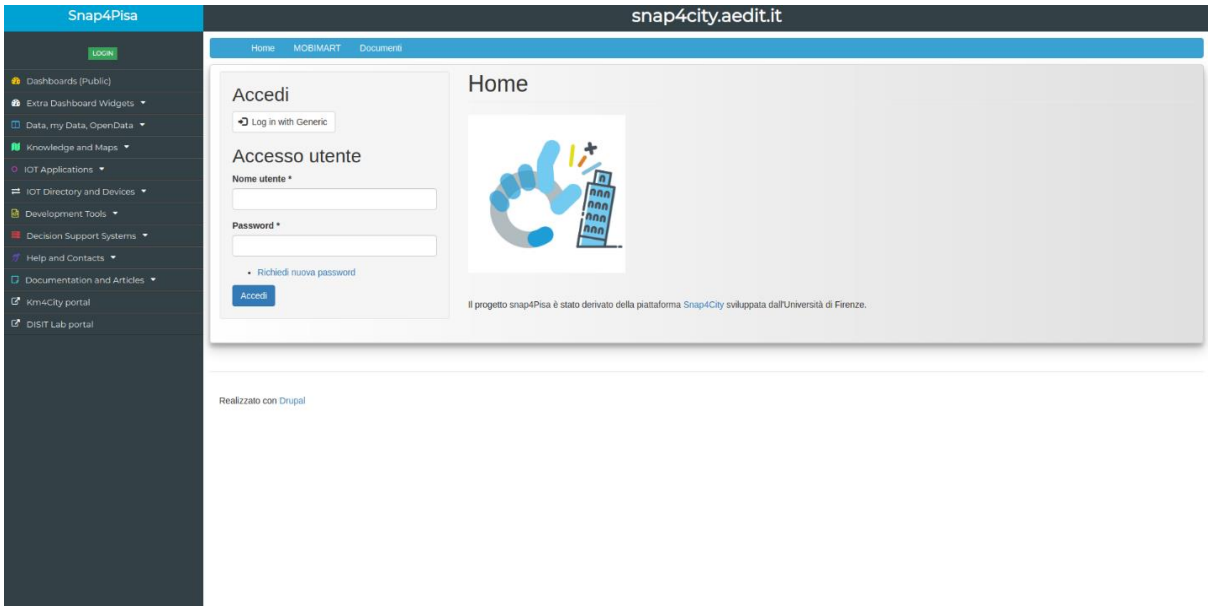
È stata creata un'istanza di drupal disponibile all'indirizzo http://snap4city.aedit.it/drupal/

Sono stati al momento installati i seguenti moduli:

- Ckeditor (consente di editare i contenuti grazie ad un'interfaccia "stile Word");
- OpenId Connect (consente l'integrazione con Keycloak per l'autenticazione).

È stata effettuata una configurazione dell'istanza di drupal e dei moduli installati in modo che sia possibile l'integrazione con il sistema di autenticazione della piattaforma snap4city.

Abbiamo creato un tema semplice basato su bootstrap, che potrà essere configurato successivamente sulla base delle indicazioni che ci verranno fornite. Sono state create le pagine di descrizione del progetto ed è stata creata una sezione per la raccolta dei documenti che saranno prodotti.



Sono state create due repository di progetto su GITHUB al seguente indirizzo https://github.com/snap4Pisa:

- **Snap4city-docker** è un fork della repository ufficiale dell'Università di Firenze sulla quale sono eseguite le personalizzazioni della piattaforma;
- **Snap4pisa_ingestion** è una repository nella quale andranno ad essere rilasciate tutte le procedure di data ingestion che saranno sviluppate nelle iotApp.

3. B.2 Caricamento dei dati

Identificazione delle possibili sorgenti di dati

In una prima fase, abbiamo identificato le possibili sorgenti dei dati che abbiamo organizzato in un file Excel disponibile in google drive al seguente indirizzo (disponibile in sola lettura):

https://docs.google.com/spreadsheets/d/12QpWf8eV6t34C1FkuIuwHnIdqf_gWNaTvNf-Re24xiU/edit#gid=696682826?usp=sharing

Sono state identificate 16 sorgenti di dati che riguardano il territorio del comune di Pisa, la persona da contattare ed altre informazioni utili. Successivamente è stata inviata una mail a ciascun contatto spiegando brevemente il progetto e invitando gli stessi ad un meeting per capire l'interesse a fornire un tipo di dato, le modalità, la frequenza di aggiornamento nonché la struttura del dato da importare nella piattaforma.
Nella seguente tabella riassuntiva sono riportate le sorgenti dati identificate.

Dati Gestiti	**Contatto**	**Stato**
People Mover	Igor.Terzariol@leitner.com	In attesa
Bike sharing	g.pin@bicincitta.com	Ricevuto il webservice
Varchi ZTL	sandro.tolaini@autostrade.it	Ricevuto il webservice (blocco firewall)
Flussi di traffico dai sensori della Regione Toscana	walter.pratesi@regione.toscana.it	Ricevuto il webservice
Air Quality (stazioni ARPAT)	Sito arpat	Importati
Trasporto Pubblico locale su gomma	Paolo Nesi	Importati (microservizio della piattaforma)
Trasporto Pubblico su ferro	Paolo Nesi	Importati (microservizio della piattaforma)
PoI	OpenstreetMap	Importati
Impianti semaforici	Massimo.Imparato@eng.it	In attesa
Flussi di traffico a cordone	Massimo.Imparato@eng.it	In attesa
parcheggi in struttura (Piazza Carrara, Piazza S.Caterina)	Massimo.Imparato@eng.it	In attesa
Varchi lungo strada	Massimo.Imparato@eng.it	In attesa
Sistemi pagamento parcheggio via smartphone	Massimo.Imparato@eng.it	In attesa
PEBA-Piano di Eliminazione delle Barriere Architettoniche	m.lazzerini@comune.pisa.it	Nessuna risposta - In attesa
Percorsi turistici principali della città	l.paoli@comune.pisa.it	Nessuna risposta - In attesa

Rilevazione in tempo reale orari aerei	Marco.Galli@toscana-aeroporti.com	Nessuna risposta - In attesa

Come si evince dalla tabella, non tutti i fornitori hanno risposto alle mail inviate nel corso dei mesi di Agosto e Settembre.

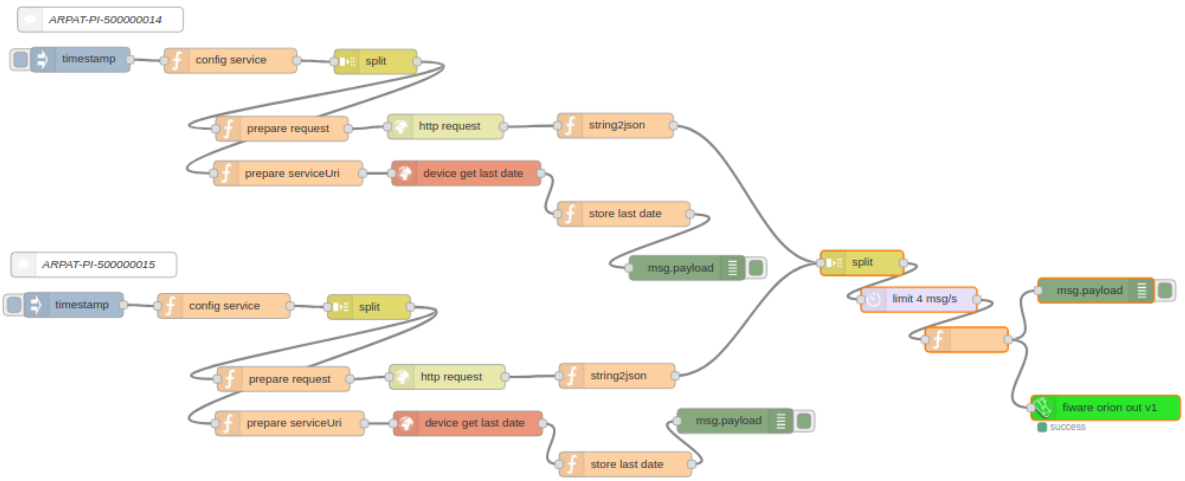
Sviluppo delle procedure di import e caricamento dei dati

Nel corso del periodo in oggetto, abbiamo iniziato a prendere confidenza con la tecnologia Node-RED utilizzata dalla piattaforma per l'import dei dati. Node-RED è uno strumento nato per la gestione del mondo dell'IoT tramite il paradigma dei flussi di dati. La programmazione di una iotApp viene fatta per blocchetti; i blocchetti possono contenere azioni automatiche o personalizzabili attraverso la scrittura di semplice codice javascript.

Sì è provveduto a scrivere le procedure generiche per l'import delle diverse tipologie al momento disponibili e che potranno essere inserite successivamente all'interno della nostra piattaforma.

Di seguito le procedure (sotto forma di screeenshot) di data ingestion sviluppate.

Dati Arpat



Il caricamento dei dati avviene automaticamente attraverso l'impostazione di un inject ripetuto a seconda della tipologia di dato per ogni intervallo di tempo. Le varie procedure che verranno realizzate saranno rilasciate all'interno della repository snap4pisa_ingestion.

Relativamente alla documentazione, è stata creata una sezione apposita all'interno del sito Drupal dove saranno man mano caricati tutti i documenti che verranno prodotti nel corso del progetto.

Abbiamo creato delle piccole guide che potranno essere modificate in futuro nel caso si rendesse necessario.

Le guide riguardano i seguenti argomenti:

- Guida a Nodered e alle iotApp;
- Guida alla creazione dei Sensori;
- Guida alla creazione di una dashboard;
- Guida all'installazione del sistema.

Progetto specifico di deploy di Snap4Pisa

È stata progettata, di comune accordo con l'Università di Firenze, la virtual machine che avrebbe ospitato tutto la piattaforma snap4city. Siamo partiti dal definire i fabbisogni hardware e software.

La piattaforma è costituita da una serie di docker connessi tra di loro e che costituiscono i servizi disponibili.

Docker è un progetto open source che utilizza le funzionalità di isolamento delle risorse del kernel Linux per consentire a "container" indipendenti di coesistere sulla stessa istanza di Linux, evitando l'installazione e la manutenzione di una macchina virtuale. Un container docker a differenza di una virtual machine, non include un sistema operativo separato. Al contrario, utilizza le funzionalità del kernel e sfrutta l'isolamento delle risorse per isolare ciò che l'applicazione può vedere del sistema operativo.

Per far coesistere una serie di container in una virtual machine, abbiamo creato quest'ultima con 100GB di spazio disco e con 32GB di memoria. Queste caratteristiche sono da considerare minime e possono essere aumentate (manualmente) nel caso si rendesse necessario.

Successivamente alla creazione della virtual machine, abbiamo individuato con l'ausilio del supporto di UniFI, la versione della piattaforma che sarebbe stata installata e configurata.

È stata scelta la configurazione DataCity small che consente di gestire una realtà quale è il comune di Pisa.

I servizi che sono stati attivati sono i seguenti:

- Dashboard (servizio principale di tutta la piattaforma che fornisce le risposte sotto forma di grafici e mappe);
- Dashboard-backend (servizio di sviluppo e personalizzazione della Dashboard);
- Dashboard-cron (servizio di automazione delle funzioni come ad esempio l'import di un dato nel tempo);
- Personaldata;
- Wsserver;
- Synoptics;
- Iotapp-nr1 (servizio di Node-RED per lo sviluppo di procedure di data ingestion);
- Iotapp-nr2 (servizio di Node-RED per lo sviluppo di procedure di data ingestion);
- Iotapp-nr3 (servizio di Node-RED per lo sviluppo di procedure di data ingestion);
- Zookeeper;
- Kafka;
- Orionbrokerfilter (servizio che permette la connessione con i diversi storage presenti all'interno della piattaforma);
- Orion;
- Servicemap (servizio tomcat);
- Virtuoso-kb (servizio di storage delle "triplette" per la definizione dei punti di interesse);
- Nifi;
- Elasticsearch (servizio di storage);
- Kibana (servizio di storage);
- Dashboarddb (servizio di storage);

- Ldap-server;
- Keycloak (servizio di autenticazione degli utenti);
- Drupal (Content Manager System per la gestione dei contenuti).

Una volta terminata la prima installazione di tutti i servizi, abbiamo testato il funzionamento di tutte le componenti del sistema in un dominio interno. Successivamente sono state identificate tutte le variabili di ambiente da cambiare per far sì che il sito funzionasse su un dominio esterno. È stato creato un file di scripting che cambia tutte le variabili con lo scopo di semplificare una possibile modifica futura come ad esempio il cambio del dominio.
Operando in stretta collaborazione con il supporto, abbiamo raggiunto l'approvazione a livello tecnico della modalità di installazione e configurazione di tutto l'ambiente.
In questo periodo abbiamo inoltre iniziato l'ottimizzazione della piattaforma attivando ad esempio l'https e iniziato la migrazione degli script testati sulla piattaforma in esercizio presso l'Università di Firenze (https://www.snap4city.org/).

4. B3 Procedure di trasformazione ed elaborazione dati

Sono stati sviluppati una serie di script open source realizzati con Node-RED che permettono l'estrazione del dato, la trasformazione dello stesso nel formato specifico in uso alla piattaforma ed infine il caricamento dei dati all'interno della piattaforma Snap4Pisa.
I dati attualmente presenti nella piattaforma sono di due macro-tipi:

- Dati statici
- Dati continui in realtime.

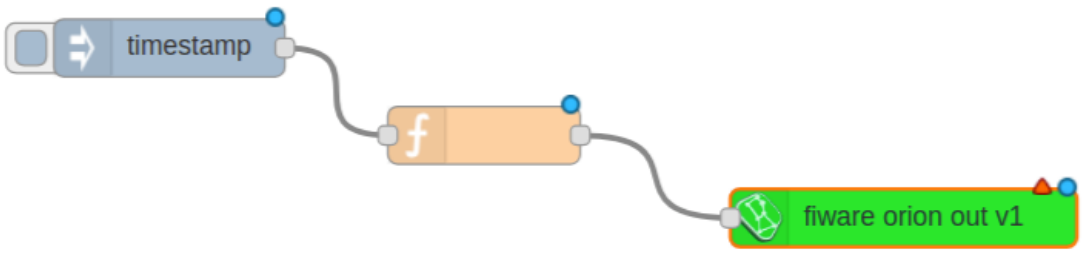
I **Dati statici** sono i punti di interesse (PoI) presenti nel territorio di Pisa suddivisi per tipologia; in essi troviamo ad esempio: Bar, Ristoranti, Parcheggi, Bancomat, Ospedali, ect.

I **Dati continui** in realtime, sono i dati provenienti da sistemi Internet of Thing (IoT). Tra questi al momento sono stati importati i seguenti dati:

- Sensori di Inquinamenti (Arpat)
- Stalli BiciInCittà
- Sensori del Traffico - Regione Toscana
- Sensori ZTL Comune di Pisa

Le procedure di data ingestion che estraggono, trasformano e caricano i dati sono come detto sviluppati utilizzando la tecnologia Node-RED che permette una programmazione a blocchi, dove ogni blocco può essere personalizzato a seconda delle esigenze del caso che si sta trattando.

A titolo esemplificativo, per poter inserire un dato servono almeno tre blocchi come mostrato nell'immagine qui sotto:
Inject - blocco che permette di dare via alla procedura; può anche essere temporizzato per essere eseguito automaticamente ogni "unità di tempo" che definiamo atomica per quel tipo di dato;
Funzione - blocco che permette di scrivere codice javascript e preparare un dato che sarà caricato nel database;
Fiware orion out v1 - blocco che permette di inserire i dati in un contextBroker o database.



Nel blocco della funzione può essere inserito del codice javascript che definisce il formato con il quale i dati saranno memorizzati all'interno del database. Ad esempio, questo breve codice crea una nuova misura rilevata dal sensore "sensore_weather" con i parametri valorizzati di temperatura e umidità.

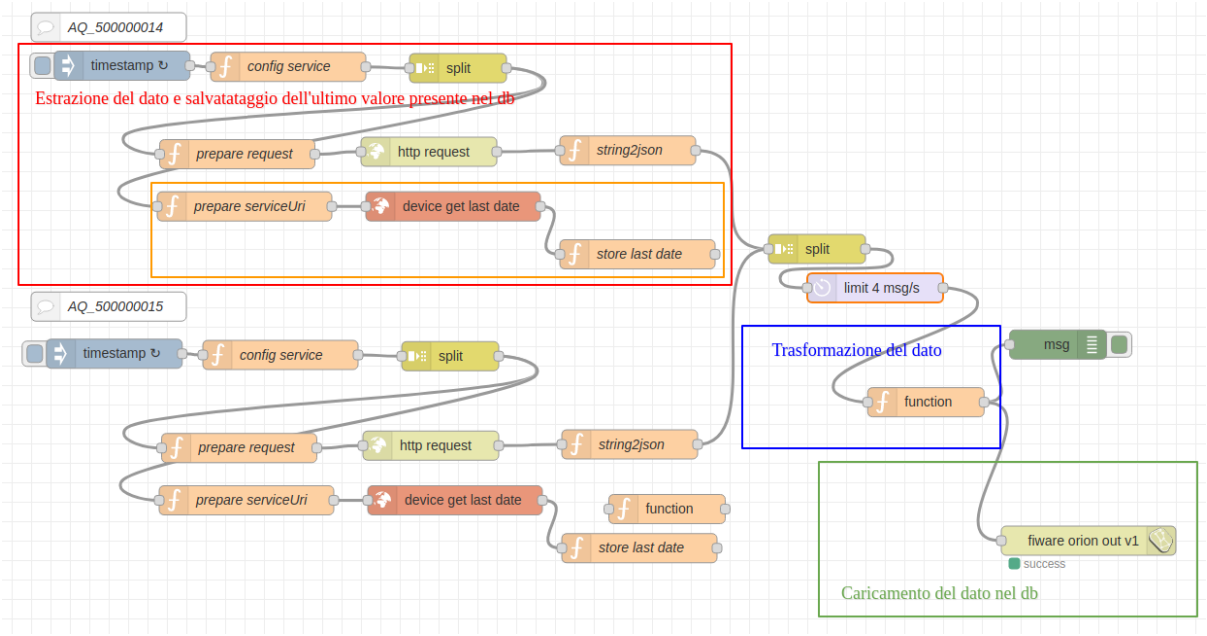
```
var data={
 "id": "sensore_weather",
 "type": "misura",
 "attributes":[
    {"name": "dateObserved", "value": new Date().toISOString(), "type":
"time"},
    {"name": "temperature", "value": 32.1 , "type": "float"},
    {"name": "humidity", "value": 44.1 , "type": "float"}
  ]
}
var msg={payload: data}
return msg;
```

I dati caricati all'interno della piattaforma secondo questa struttura, potranno poi essere inviati alle dashboard dove saranno mostrati nelle diverse forme definite dai widget disponibili; questo aspetto del sistema sarà descritto nella sezione B4 più avanti in questo report.

B3.1 Import dati Arpat

Sono stati creati due flussi, uno per ogni sensore presente nel comune di Pisa, con un'ultima parte in comune che si occupa del caricamento finale nel database.

Dì seguito lo script Node-RED che esegue l'estrazione in rosso, la trasformazione in blu attraverso uno script javascript e infine il caricamento del dato in verde:



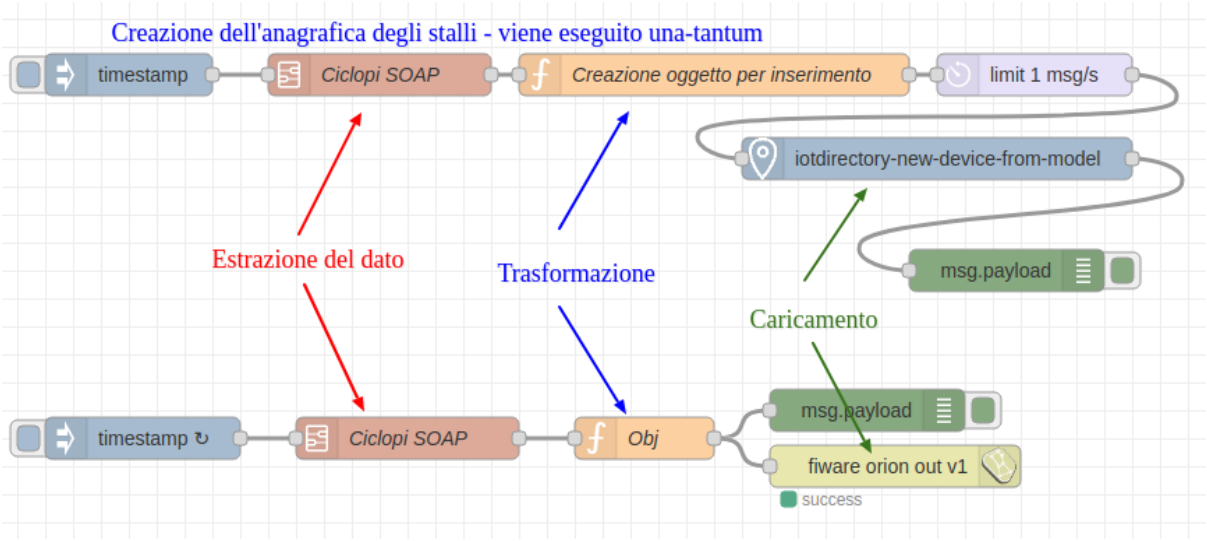
Lo script, al fine di limitare la duplicazione dei dati, memorizza l'ultimo valore inserito (sottosistema arancio) in modo da inserire solo i nuovi valori rilevati dal sensore.

Dato	Sensori Arpat
Step del dato	Orario
Ingest	Ogni ora

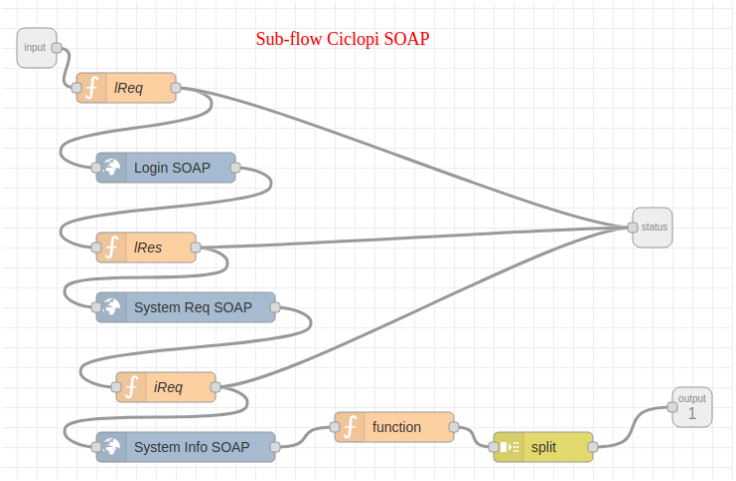
B3.2 Stalli BiciInCittà

Sono stati creati due flussi, uno per inserire le anagrafiche di ciascun stallo e l'altro per inserire i dati sulla disponibilità delle biciclette all'interno dello stallo.

Dì seguito lo script Node-RED che esegue l'estrazione in rosso, la trasformazione in blu attraverso uno script javascript e infine il caricamento del dato in verde:



In questo caso è stato creato un sub-flow che esegue una serie di operazioni con lo specifico fine di estrarre il dato. L'immagine con i blocchetti che si occupano dell'estrazione dei dati sono mostrate nella prossima immagine:

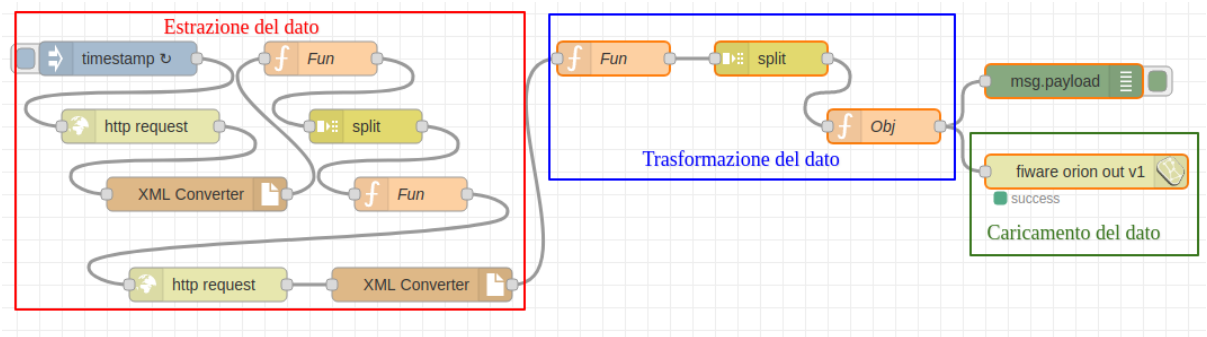


Dato	Sensori BikeSharing
Step del dato	Real time
Ingest	Ogni 5 minuti

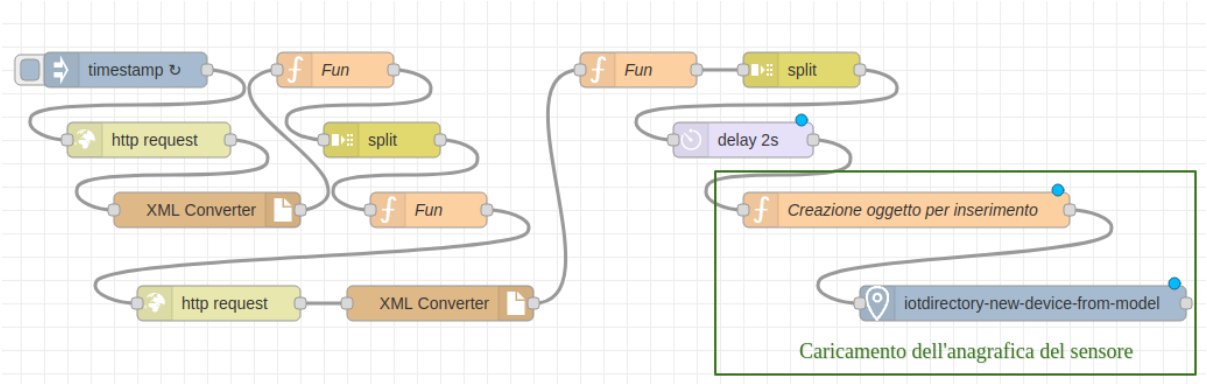
B3.3 Sensori di traffico

Anche in questo caso sono stati creati due flussi, uno per inserire le anagrafiche di ciascun sensore di traffico e l'altro per inserire i dati dei passaggi dei veicoli.

Dì seguito lo script Node-RED che esegue l'estrazione in rosso, la trasformazione in blu attraverso uno script javascript e infine il caricamento del dato in verde:



Il secondo flusso viene utilizzato una tantum per inserire l'anagrafica dei vari sensori di traffico installati nel territorio del comune di Pisa.



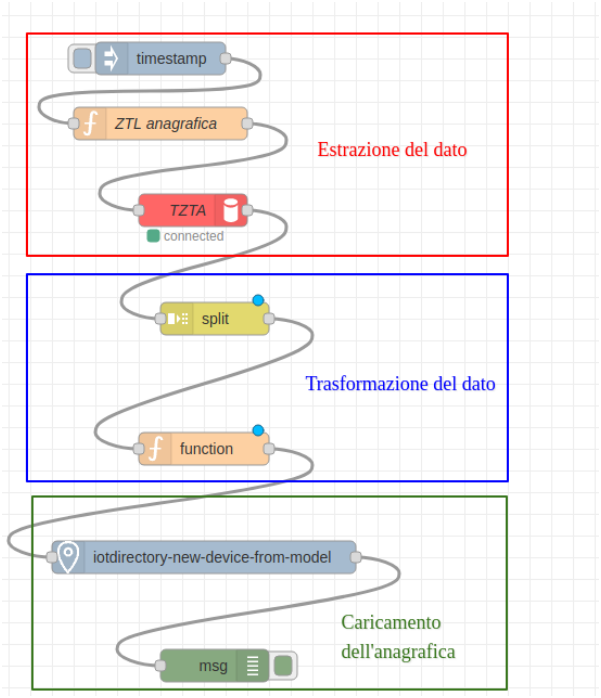
Dato	Sensori Flussi di Traffico
Step del dato	Real time
Ingest	Ogni 3 minuti

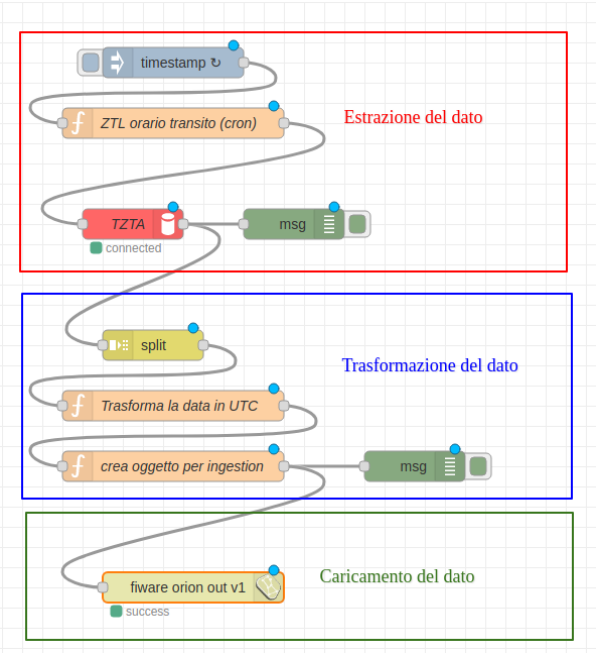
B3.4 Sensori ZTL Comune di Pisa

Anche in questo caso sono stati creati due flussi, uno per inserire le anagrafiche di ciascun sensore ZTL e l'altro per inserire i dati delle rilevazioni dei veicoli.

Dì seguito i due script Node-RED: il primo che esegue l'estrazione in rosso, la trasformazione in blu attraverso uno script javascript e infine il caricamento del dato in

verde relativo alla anagrafica della telecamera ZTL mentre il secondo inserisce il dato del passaggio in realtime.





Dato	Sensori ZTL
Step del dato	Real time
Ingest	Ogni 2 minuti

B3.5 Parcheggi

In questo caso è stato realizzato uno script esterno alla piattaforma, sviluppato in python. Dopo aver configurato tutte le telecamere ad inviare le foto delle rilevazioni, è stata creata una app python Flash che interroga tutte le immagini al fine di estrapolare i dati di interesse. Di seguito lo script

```
from glob import glob
from os import rename, listdir, path, remove
from aeCommon.MY import MY as DB
from config import DBMYSQL, INPUT_FOLDER


db = DB(DBMYSQL)


dir = [ name for name in listdir(INPUT_FOLDER) if path.isdir(path.join(INPUT_FOLDER, name)) ]
fields = ['S_DATE', 'S_TIME', 'S_PLATE', 'S_CLASS_STRING', 'I_FILENAME', 'I_LANE', 'I_DIRECTION']
```

```
for d in dir:
    workfolder = INPUT_FOLDER + d + "/"
    files = glob(workfolder + "*.jpg")
    # print d, files

    for img in files:
        print img

        try:
            if path.getsize(img) > 0:
                f = open(img,"r")

                filename = ""
                name = ""
                ins_field = ""
                ins_value = ""
                val = {}
                for line in f:

                    for fld in fields:
                        if fld in line:
                            line = line.replace("\r","").replace("\n", "")
                            aFld = line.split("=")

                            field = aFld[0].lower()
                            value = aFld[1]

                            if fld == "S_TIME":
                                value = value.replace('-', ':')

                            ins_field = ins_field + field + ", "
                            ins_value = ins_value + "%(" + field + ")s, "
                            val[aFld[0].lower()] = value

                            if fld == "I_FILENAME":
                                filename = value
                                name = filename[:12]

                # ins_field = ins_field[:-2]
                # ins_value = ins_value[:-2]

                ins_field = ins_field + 'name'
                ins_value = ins_value + '%(name)s'
                val['name'] = name

                sql = "INSERT INTO log ("+ins_field+") VALUES ("+ins_value+")"
                # print sql, val
                ret = db.insert(sql,val)
                print(ret)
```

Infine è stato configurato un il cron per essere eseguito ogni minuto

Dato	Parcheggi
Step del dato	Real time
Ingest	Ogni 1 minuto

5. B4 Cruscotti e dashboard

In questa fase di realizzazione del progetto, abbiamo iniziato a sviluppare le prime dashboard per la visualizzazione dei dati. Le dashboard sono uno strumento pensato per la visualizzazione dei dati attraverso dei widget preconfezionati.

Le dashboard attualmente presenti nel sistema costituiscono un primo approccio allo strumento teso a capirne le potenzialità in attesa di realizzare delle personalizzazioni delle stesse sulla base dei feedback raccolti nella riunione di co-creazione e co-progettazione che sì stanno tenendo in questi giorni insieme agli stakeholders invitati a partecipare.

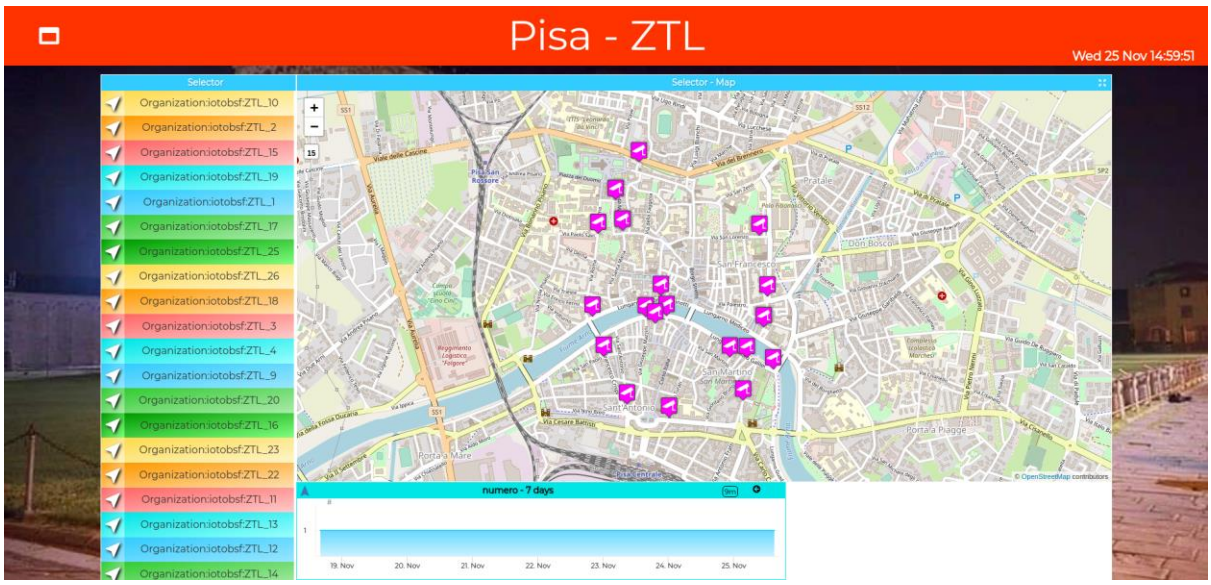
B4.1 Dashboard inquinanti

La dashboard qui di seguito mostra il trend della qualità dell'aria rilevata dalle due centraline che sono presenti all'interno del Comune di Pisa. Per ogni paramentro NO2, O3 e CO sono presenti tre widget:

- Comparazione giornaliera
- Comparazione settimanale
- Ultimo valore rilevato

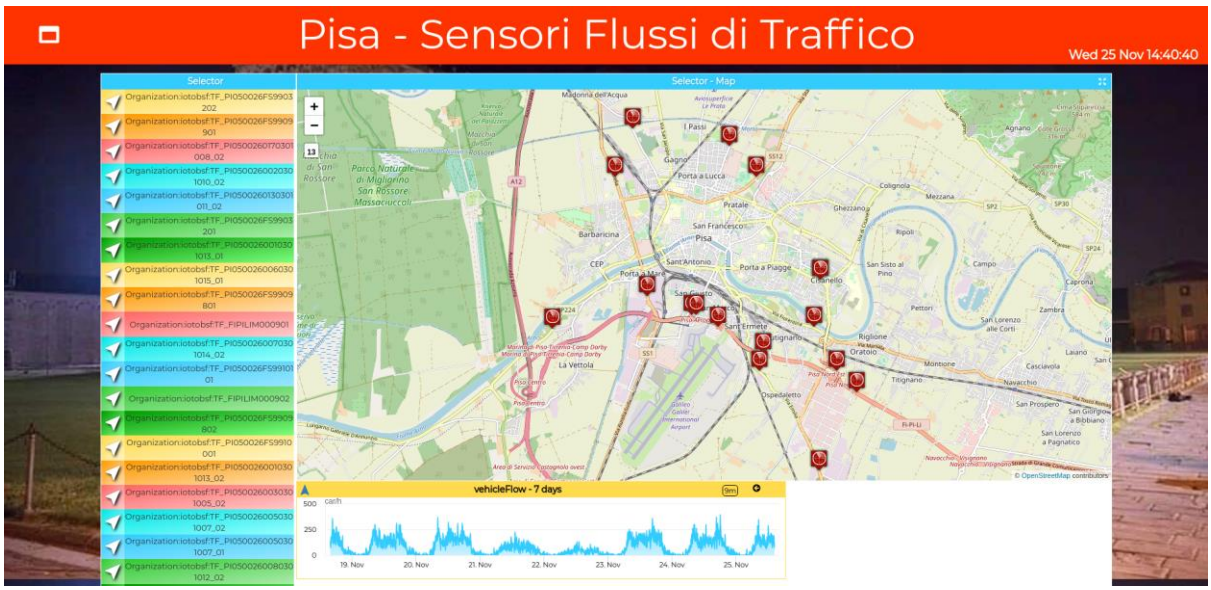
B4.2 Dashboard ZTL

La dashboard qui di seguito mostra la posizione delle telecamere della ZTL presenti sul territorio del Comune di Pisa.



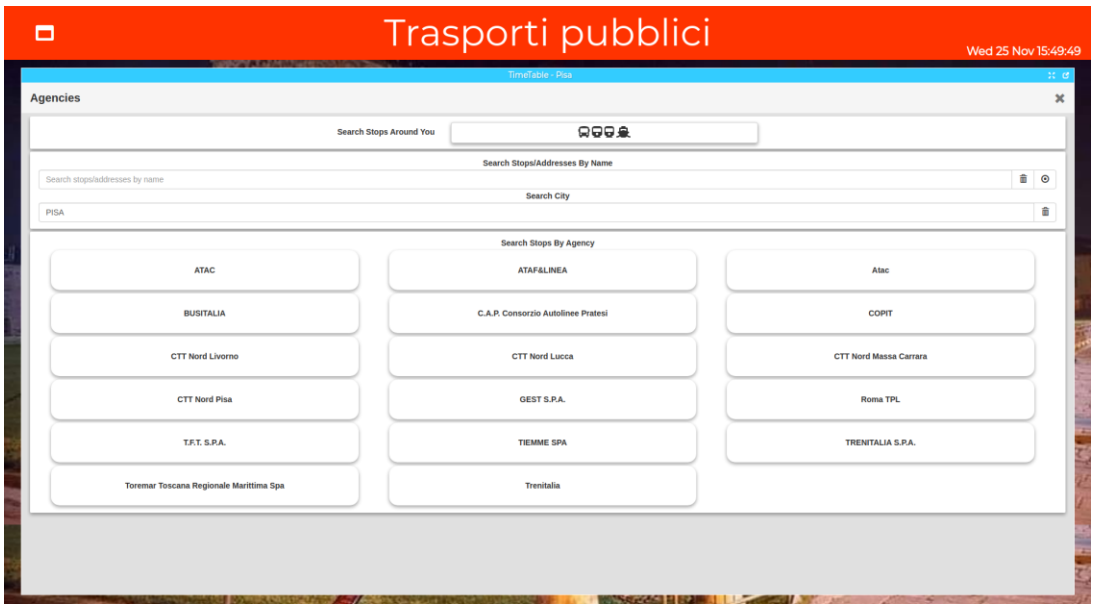
B4.3 Dashboard Flussi di traffico

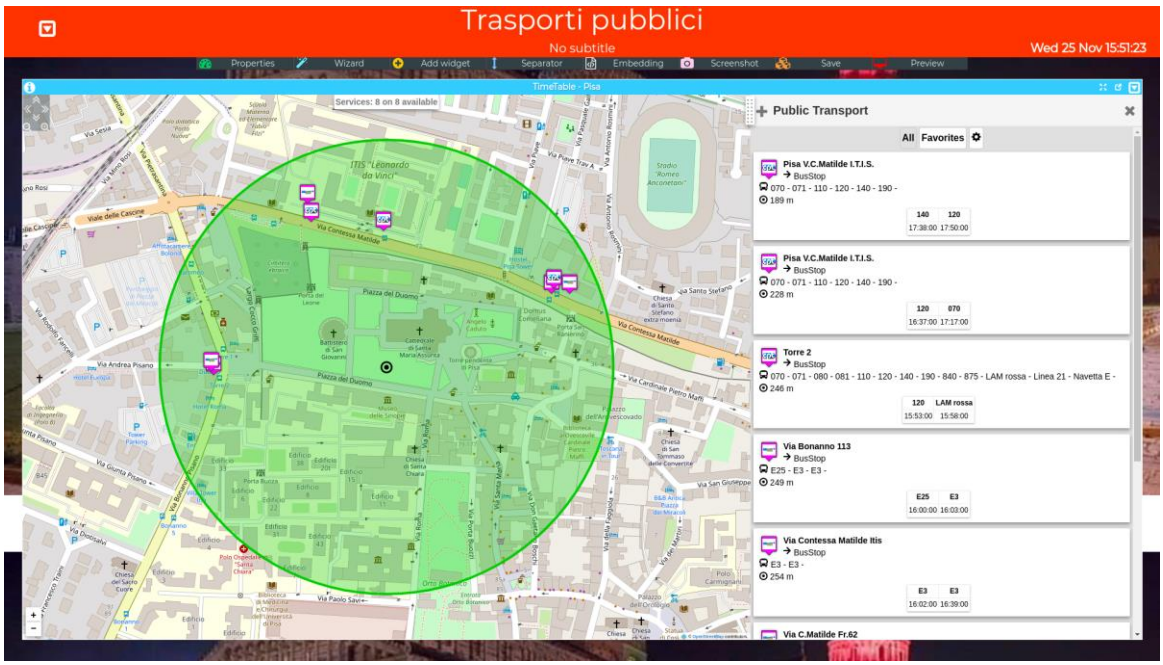
La dashboard qui di seguito mostra la posizione dei sensori di rilevazione dei flussi di traffico presenti sul territorio del Comune di Pisa.



B4.4 Trasporti pubblici

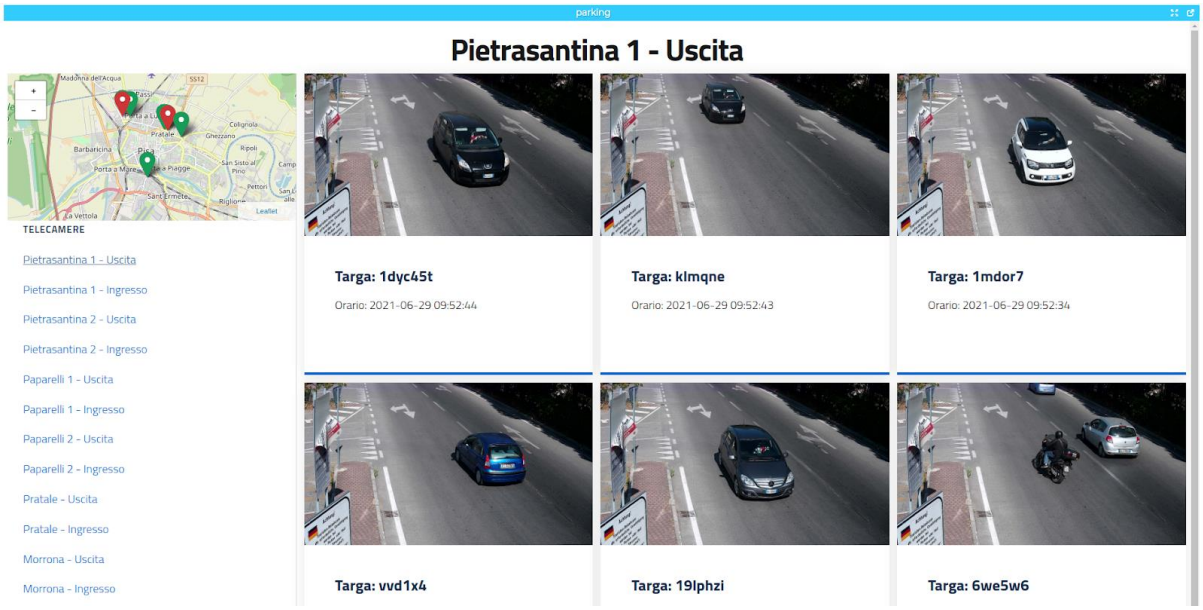
La dashboard qui di seguito mostra i trasporti pubblici. È possibile ricercare un servizio sulla base dell'agenzia di fornitura del servizio (sono presenti le agenzie operanti in tutta la Toscana), oppure è possibile ricercare la fermata più vicina attraverso la posizione GPS del dispositivo. Le due immagini di seguito mostrano queste due tipologie.



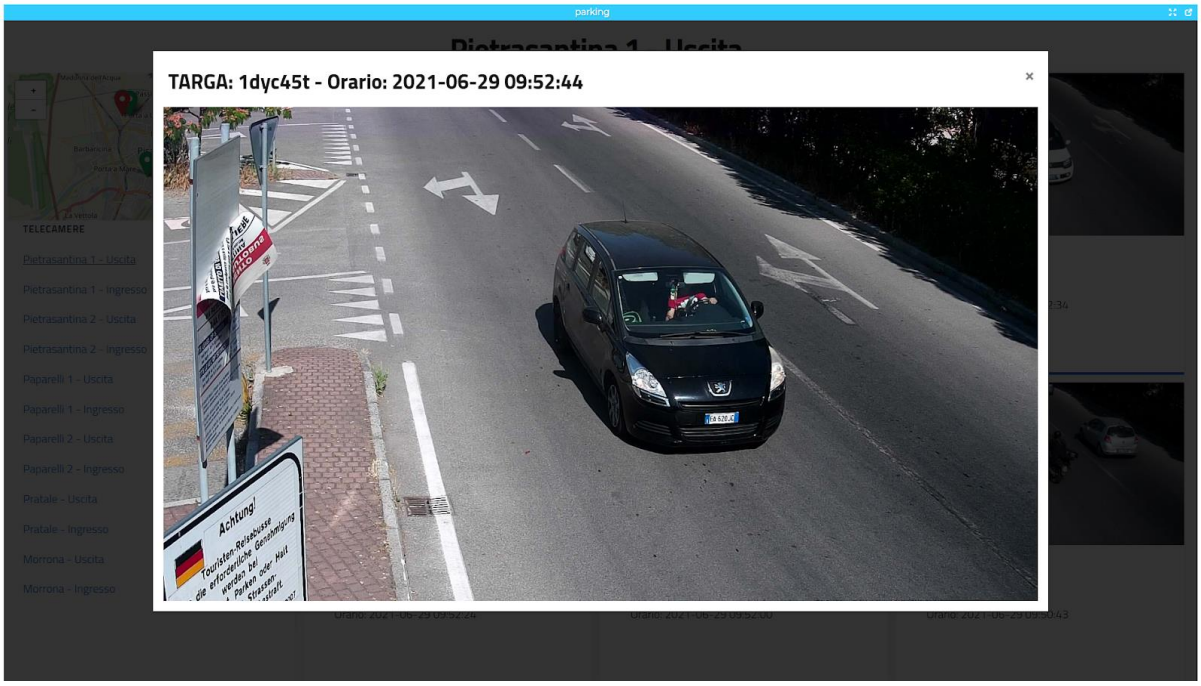


B4.5 Parcheggi

La dashboard qui di seguito mostra gli accessi ai parcheggi che sono stati installati e configurati nel corso del progetto.



La dashboard mostra una mappa con il posizionamento delle telecamere nei parcheggi. I marker sono rappresentati in due colori per identificare le telecamere di ingresso al parcheggio da quella di uscita. Esistono poi una serie di link alle telecamere; cliccando su ognuno di questi sì accede alla telecamera corrispondente. La parte destra della dashboard mostra le ultime 12 rilevazioni effettuate. Ogni rilevazione riporta la fotografia del mezzo insieme alla targa e all'orario di ingresso/uscita. Cliccando su una rilevazione il sistema mostrerà lo zoom come mostrato nella seguente immagine.

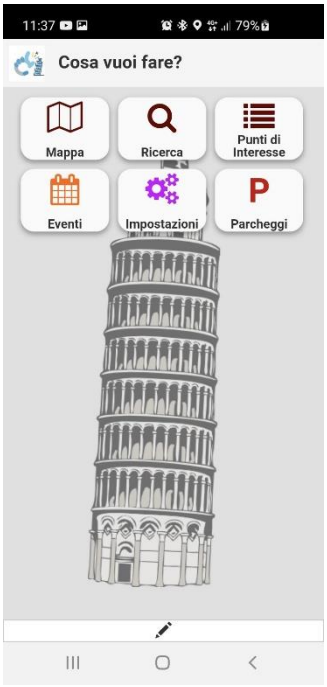


6. B5 Creazione dell'App Pisa in a SNAP

A partire da Febbraio abbiamo iniziato a lavorare su una versione prototipale dell'app partendo da uno starter kit messo a disposizione dall'Università di Firenze attraverso la repository https://github.com/disit/snap4cityAppKit L'app è sviluppata con il framework cordova che rende possibile l'utilizzo dello stesso codice sorgente anche per la generazione di una versione per l'ambiente apple. Attualmente l'app non è stata caricata sullo store Android ma è disponibile al seguente link: https://snap4city.aedit.it/drupal//sites/default/files/snap.apk

Di seguito alcuni screenshot dell'app.

Splash screen dell'app



Menu principale dell'app

Esempio di ricerca dei servizi vicini alla mia posizione

Esempio di ricerca dei servizi in base alla tipologia richiesta

L'app può essere successivamente sviluppata aggiungendo nuovi bottoni nel menu principale.

7. B7 Corsi di formazione ed aggiornamento

Per tutta la durata del progetto siamo stati in contatto con l'Università di Firenze e in particolare con il prof. Nesi, responsabile della piattaforma snap4city che con l'aiuto dei propri collaboratori ci ha supportato in gran parte delle nostre richieste.

Al fine di minimizzare il tempo di intervento abbiamo convenuto nella creazione di un gruppo di lavoro su piattaforma Skype. La chat sempre attiva ha dato modo a tutti la conoscenza degli argomenti di discussione e la possibilità di intervenire.

All'inizio del progetto sono state organizzate delle sessioni di introduzione alla piattaforma attraverso le quali, il team di Aedit, ha potuto organizzare dei piccoli task di lavoro al fine di mettere in pratica ciò che veniva spiegato. Dalla messa in pratica potevano nascere necessità particolare che sono state trattate o attraverso domande/risposte sulla chat di gruppo o attraverso una call che andava ad analizzare e risolvere il problema specifico.

8. B8 Formazione

Per tutta la durata del progetto sono state realizzate delle guide alla piattaforma che permettono, insieme ai tutorial redatti dall'università di Firenze, di essere autonomi.

Le quattro guide realizzate e disponibili a questo link trattano i seguenti argomenti:

- Guida a Nodered e alle iotApp
- Guida alla creazione dei Sensori
- Guida alla creazione di una dashboard
- Guida all'installazione del sistema

I tutorial possono essere visionati al seguente link e trattano i seguenti argomenti:

- Overview
- Dashboards
- IOT App, IOT Network
- Data Analytics

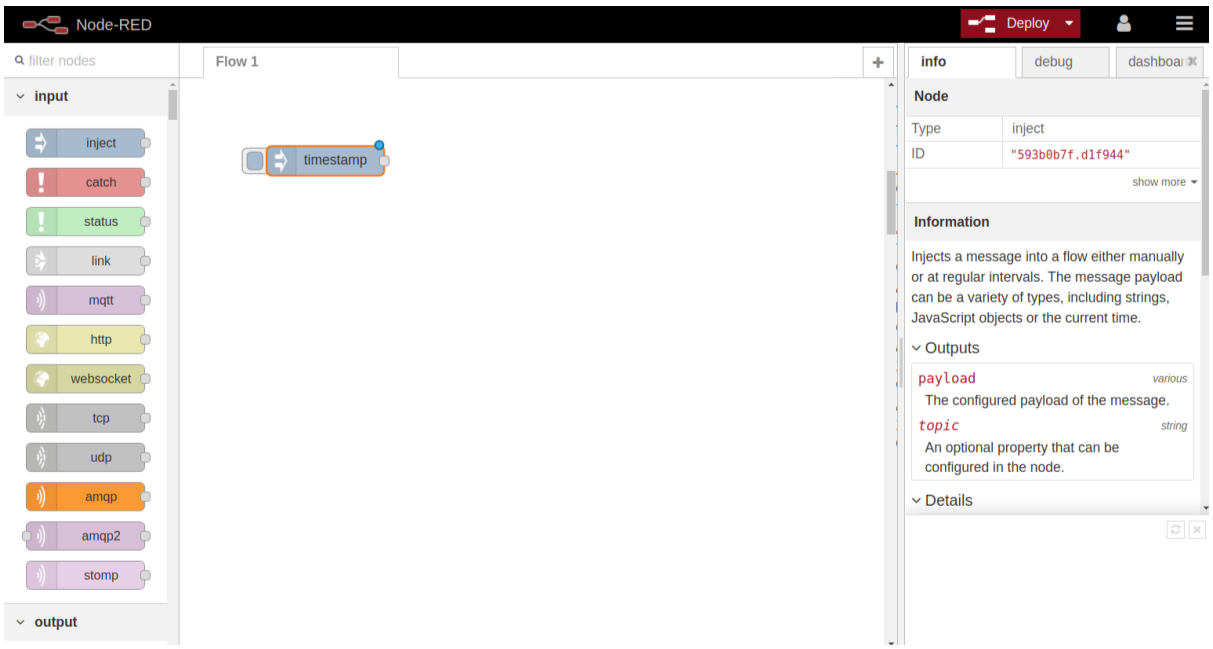
- Data Ingestion processes
- System and Deploy Install
- Smart City API: Web & Mob. App

ANNEX

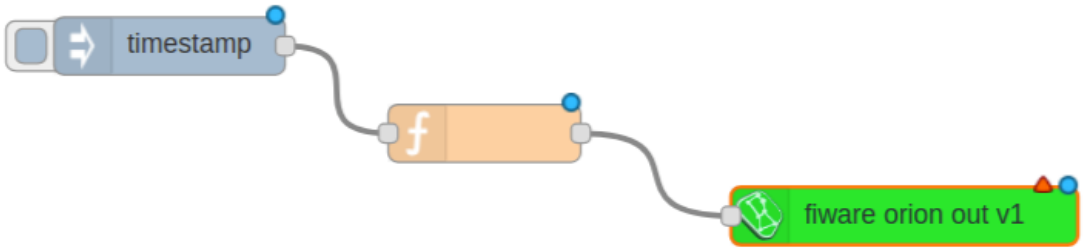
Guida base a Nodered e alle iotApp

Inserire un dato IoT con una IoTApp

- andare su IOT Applications e creare una nuova IOTApp; l'application Type deve essere basic. E' possibile anche ri-usare una IoT Application che avete già creato; si consiglia di inserire in 1 iotApp più di 1 procedura per poterle gestire in maniera più semplici e per non occupare troppe risorse del sistema
- l'iotAPP ha dei blocchi nella colonna di sinistra; selezionare il block "inject" e trascinarlo nella board; Il blocco serve per poter avere un comando per far iniziare una serie di operazioni a cascata; dopo averlo trascinato cliccare su "deploy" per "pubblicare" l'app. A questo punto se cliccate sul "pulsante" alla sinistra del blocco il sistema "inietta" (inserisce) una variabile nel sistema, in questo caso inserisce la data corrente (timestamp) (compare in messaggio "Successfully injected: timestamp)"



- per poter inserire un dato servono almeno tre blocchi; inserire i blocchi e collegarli tra di loro nel modo che si vede nell'immagine successiva:
  - inject: già visto per dare il via alla procedura
  - function: è un blocco che permette di aggiungere codice javascript
  - fiware_orion_out_v1: questo blocco permette di inserire i dati in un contextBroker di tipo "Orion Context Broker" alla versione v1; questo è il device attualmente più diffuso, nella libreria ci sono dei blocchi dedicati a contextBroker di altre versioni

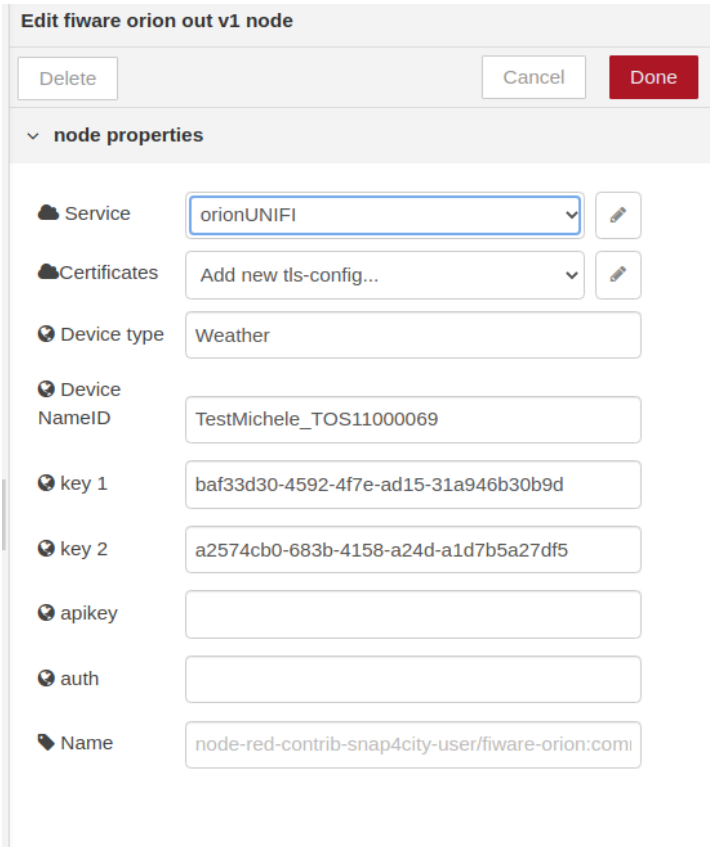


- cliccare sul blocco funzione (quello al centro) e scrivere la seguente funzione di inserimento dei dati; nell'esempio che mostriamo inseriamo ogni volta che si clicca su inject un dato con la data corrente e con dei valori fissi
  - id: il codice esatto del device che avete creato
  - type: il tipo che avete definito nel iotModel e dell'iotDevice
  - name: i nomi esatti degli attributi definiti nell'iotModel
  - value: i valori che vogliamo inserire; new Date() inserisce la data corrente

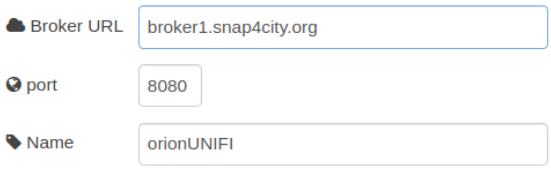
```
var data={
 "id": "test_diego_pisa",
 "type": "misura",
 "attributes":[
   {"name": "dateObserved", "value": new Date().toISOString(), "type": "time"},
   {"name": "temperature", "value": 32.1 , "type": "float"},
   {"name": "humidity", "value": 44.1 , "type": "float"}
 ]
}
var msg={payload: data}
return msg;
```

Attenzione il campo temporale (dateOserved) deve essere in formato testuale ISO: es: "2020-08-07T12:51:50.548Z"; IN JS da un campo data si ottiene la stringa ISO con la funzione toISOString();

- Configurare il Fiware orion out v1
  - fare doppio click
  - inserire orionUNIFI
  - inserire k1 e k2 (se non li avete li trovate nella scheda del device)



Dove il service va modificato con la matita inserendo **orionUNIFI** come nella seguente figura



Importante: È possibile passare l'autenticazione del sensore nel messaggio che viene inviato al Fiware orion out v1; per farlo è sufficiente preparare l'oggetto nel seguente modo:

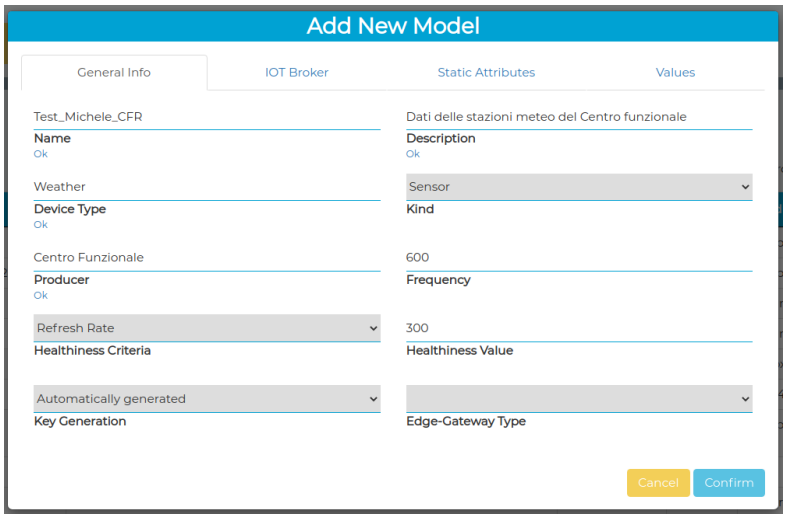
```
var k1 = "f6efd060-375b-4f40-af09-41e0814f7039";
var k2 = "108bfe91-8140-41fd-92d8-c2b317c3cef8";
...
var auth={"k1": k1,"k2": k2,"apikey":"apikey","basicAuth": "basicAuthKey"};
return {"auth": auth, "payload":ret}
```

Guida alla creazione dei Sensori

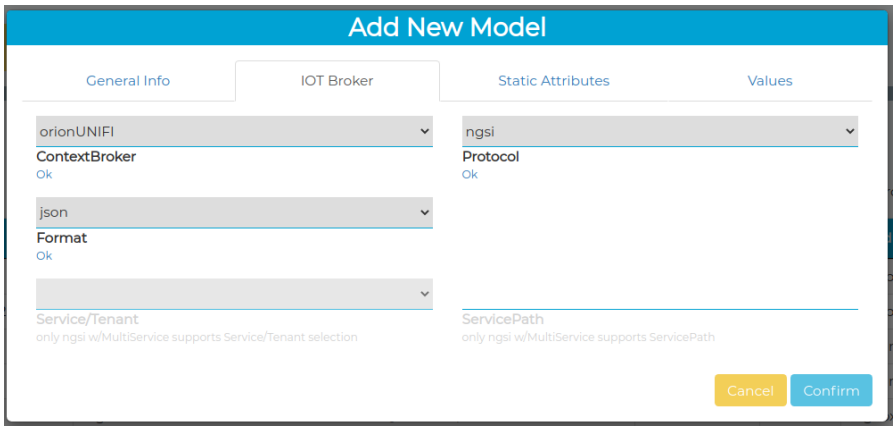
Creazione di un IoT Device Model

Nel seguente link una descrizione completa in inglese di come si inserisce e configura un https://www.snap4city.org/drupal/node/591)

- va definito il nome del device, la descrizione ed il tipo; in questo caso si inserisce il dato di una stazione meteo e quindi si usa Weather come tipologia



- cliccare su “IOT Broker” e scegliere il broker dalla tendina

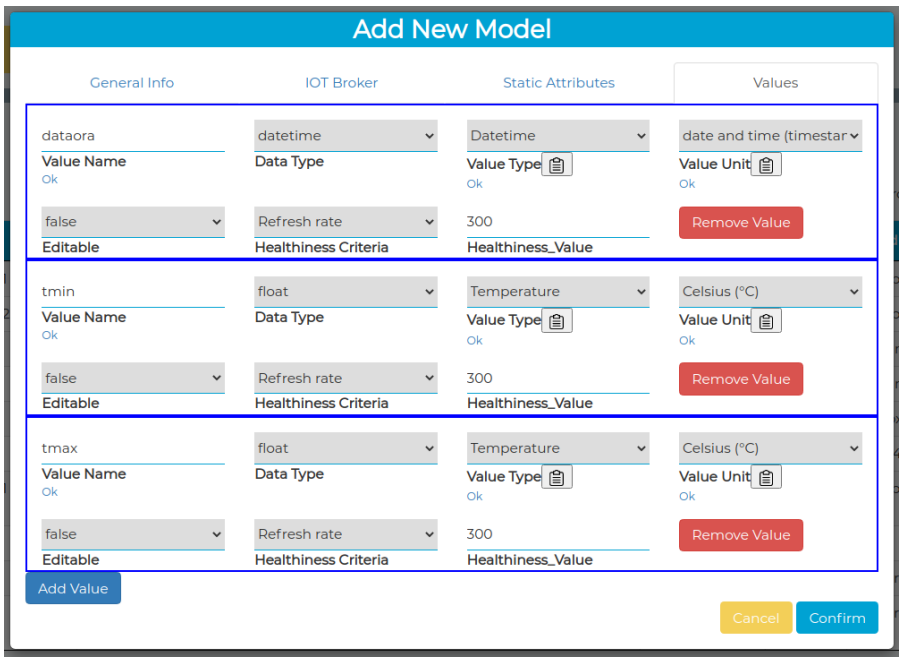


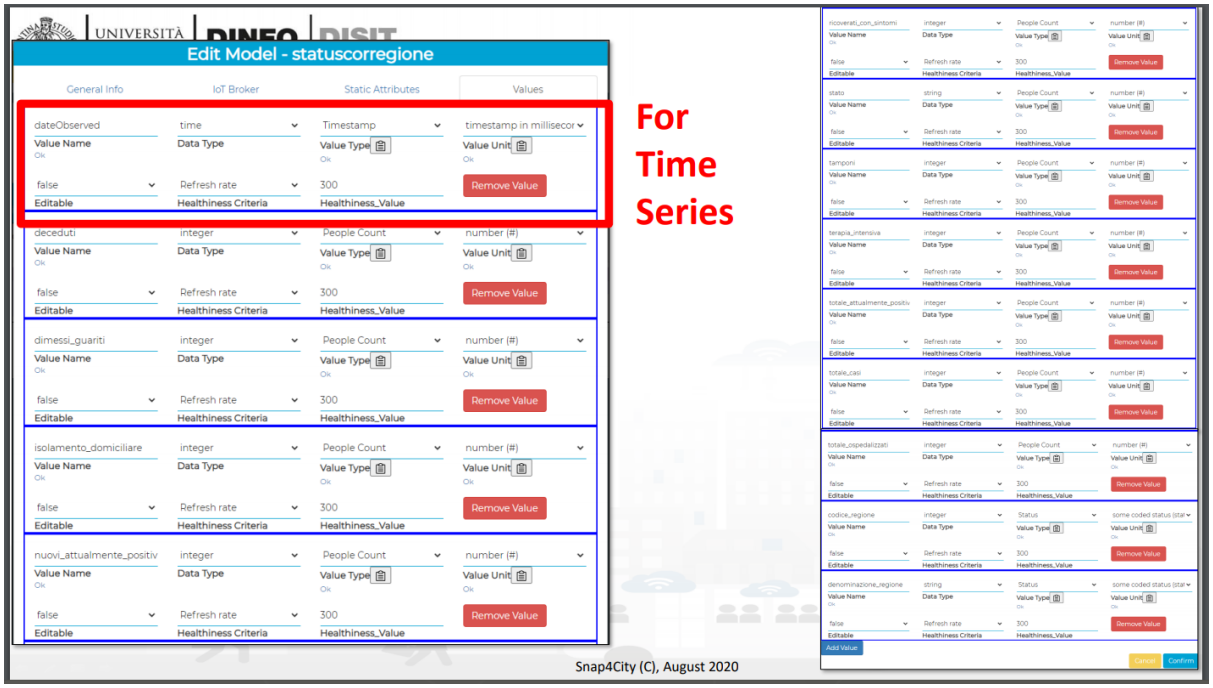
nel tab Values sì deve andare a definire la struttura dei dati che verranno acquisiti dal modello
per ogni variabile aggiungere una “riga” con Add Value
selezionare il data type (data, stringa, float, …)
selezionare dalla tendina il value type
selezionare il value unitbnn0

Attenzione!!!. Nel dataModel se si ha a che fare con una serie temporale, occorre creare un campo codificato in questo modo:

- il nome deve essere dateObserved (da controllare)
- il Data Type deve essere Time
- il Value Type deve essere **Timestamp**
- il valued deve essere timestamp in millisecond

Se non si codifica bene poi il sistema non riconosce la data di misurazione.

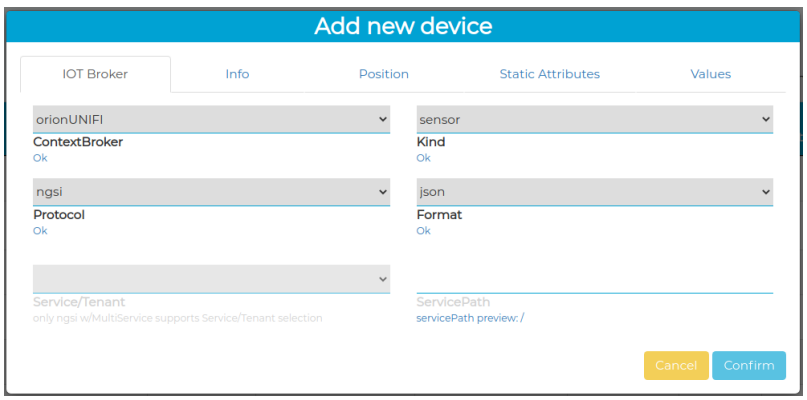




Registrazione Manuale di un IoTDevice

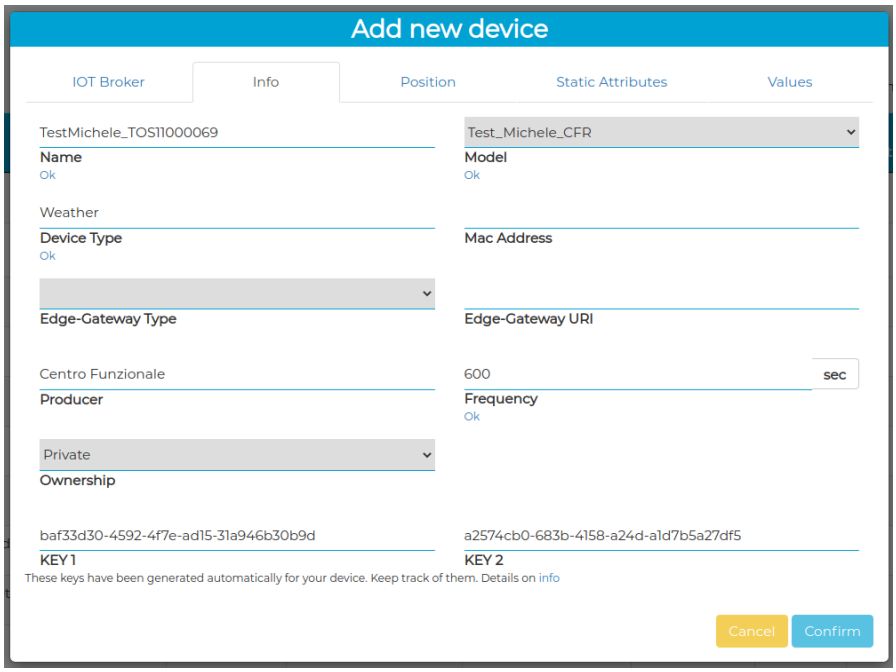
Aggiungi un nuovo IoTDevice

- selezionare il contextBroker

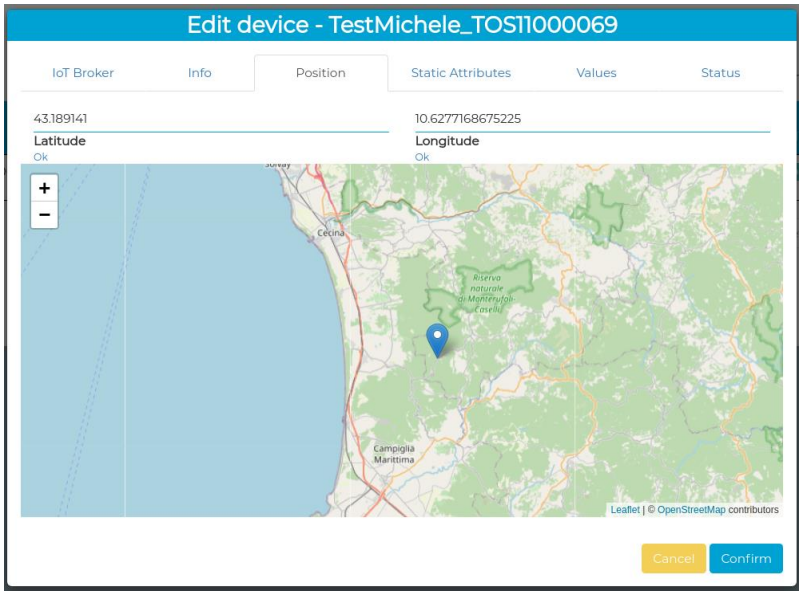


- Nel tab Info:
  - assegnare un nome

- selezionare il modello (creato al punto precedente)
- salvarsi separatamente le chiavi KEY1 e KEY2



- Nel tab Position inserire le coordinate



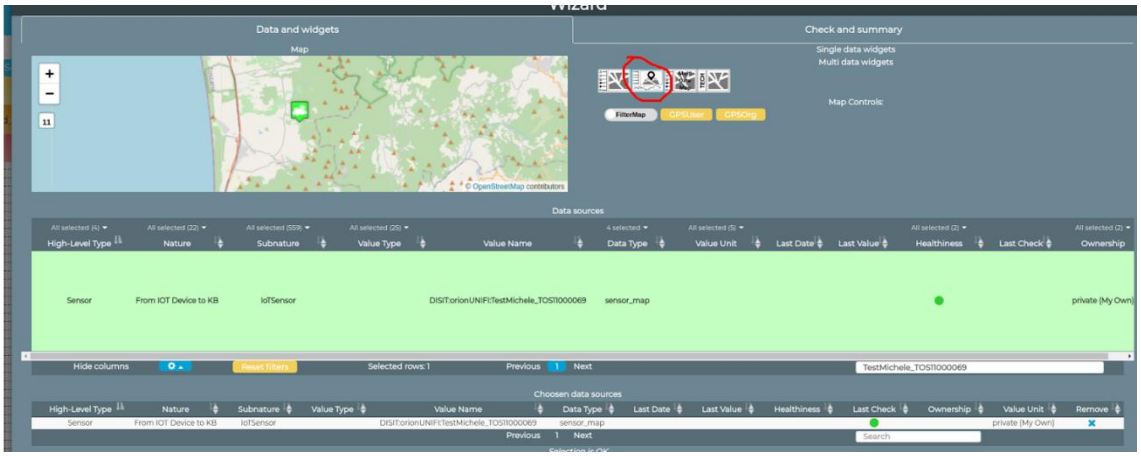
Salvare il device creato

Suggerimento: Nel caso di definizione di più sensori dello stesso tipo (in generale potremmo allargare a tutti i sensori del progetto Pisa), possiamo impostare gli stessi k1 e k2 per tutti i sensori.

Guida alla creazione delle Dashboard

Creare una dashboard / aggiungere il widget ad una dashboard esistente

- Aprire la Wizard
- Cercare il nome del device che è stato creato
- Selezionare la prima riga (quella con il pallino verde)
- Selezionare il tipo di widget evidenziato nella figura sottostante.



Guida all’installazione del sistema

Installazione di snap4city

Preparare la macchina

- la RAM deve essere di almeno 32 GB (altrimenti delle macchine possono fallire)
- Per elasticsearch è necessario configurare il max_map_count della macchina host; se non ha questo valore potrebbe avere dei problemi di memoria (il container docker viene subito fermato)
    - nano /etc/sysctl.conf
    - #aggiungere la seguente riga
    - vm.max_map_count=262144
    - sysctl -p

Scaricare il docker-composer ed installare seguendo le istruzioni

```
git clone https://github.com/disit/snap4city-docker
cd snap4city-docker/DataCity-Small

# setup directories write permissions and sets vm.max_map_count=262144 for elasticsearch
./setup.sh

#bring all services up
docker-compose up -d
#to bring all services up will takes some minutes
```

Check delle macchine e post setup

Attenzione!!!! Occorre verificare che tutte le macchine siano partite. Nello specifico soprattutto le macchine con JAVA possono dare problemi di RAM e bloccarsi. Anche ElasticSearch può avere dei problemi se il server non è configurato bene.

```
#then setup virtuoso and elasticsearch (to be done only the first time but no problem if you repeat)
./post-setup.sh
```

A questo punto seguire le istruzioni e dovrebbe funzionare in localhost.

Installazione in ambiente di esercizio

Per poterlo installare in esercizio occorre assegnare alla macchina un dominio e poi modificare in tutti gli script il nome del dominio stesso. Vanno modificati anche i riferimenti ai menu nel database mySQL. Abbiamo realizzato uno script **snap4city_onpremise_setup.sh** che effettua un backup dei soli file che verranno modificati con il nome del dominio che daremo all'ambiente di esercizio (nel nostro caso snap4city.aedit.it)

Dopo il run dello script vanno cambiati i redirectURI che sono in keycloak.

- keycloack:
  - accedere al sito con le credenziali admin / admin
  - andare nel clients
  - modificare i seguenti Clients
    - Js-grp-client
    - Js-kpi-client
    - Js-synoptic-client
    - Nodered
    - Php-dashboard-builder
    - Php-iot-directory
    - Php-notificator
    - Php-ownership-api
  - per ognuno aggiungere un Redirect URIs = <nome_del_dominio>

Modifiche al docker-compose

- aggiungere un volume a keystore per non dover perdere ad ogni restart la configurazione

Nel docker-compose aggiungere il volume:

```
keycloak:
    ...
    volumes:
     - ./keycloak-db:/opt/jboss/keycloak/standalone/data
```

Va “salvata” la configurazione di snap4city nel volume che sì sta creando attraverso il comando da eseguire all’interno del container

> docker exec -it datacitysmall_keycloak_1 bash
> scp -r /opt/jboss/keycloak/standalone/data/* aedit@192.168.213.68:/opt/snap4city-docker/DataCity-Small/aedit/keycloak-db