Interreg

UNION EUROPÉENNE
UNIONE EUROPEA

MOBI MART

MARITTIMO–IT FR–MARITIME

Fonds européen de développement régional
Fondo Europeo di Sviluppo Regionale

PROJET MOBIMART
ID prodotto T3.2.2
Module spécifique à la plate-forme

Mars 2021 - Pisa

REGIONE TOSCANA
REGIONE LIGURIA
REGIONE AUTÒNOMA DE SARDIGNA
REGIONE AUTONOMA DELLA SARDEGNA
RÉGION SUD PROVENCE ALPES CÔTE D'AZUR
COLLECTIVITÉ DE 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

Projet numéro	168
Acronyme	MOBIMART
Titre du projet	Mobilità intelligente mare terra
Début/fin du projet	01.04.2018- 30.06.2021 + 120 gg
Durée	39 + 120 gg

Élement	T3 Systèmes d'information provinciaux et municipaux
Activité à laquelle le produit se réfère	T 3.2 Système d'information sur les transports et la mobilité
Titre du produit	Module spécifique à la plate-forme
Stage de référence	P6
Période de référence (Début/Fin)	01/10/2020 – 31/03/2021

Auteur	**AEDIT s.r.l.** Via Fornace Braccini 8 – 56025 Pontedera (PI) P.IVA: 01814780464 Pisamo Srl Comune di Pisa
Version	1.1
Date	23/03/2021
Responsable de la validation	Comune di Pisa
Date de révision	
Sommaire de modifications	

Auteur des changements	

Indice/Index

1. Introduction

Ce document contient la description des activités menées dans le cadre du projet du 1er août 2020 au 31 mars 2021. Après s'être familiarisé avec la plateforme Snap4city, nous avons d'abord développé des procédures pour le data ingestion et pour la création de Dashboard pour afficher les données précédemment chargées. Une Virtual Machine (VM) a été créé sur notre serveur où la dernière version disponible de la plateforme était déjà installée. Nous avons progressivement ajouté des sources de données qui ont été régularisées grâce à l'exécution de procédures chronométrées. Dans le même temps, nous avons lancé des activités de visualisation de données à travers le développement de la dashboard spécifiques pour les types de données.

Pendant toute la durée de la période, nous avons été en contact avec le soutien de l'Université de Florence qui a réalisé des activités de formation au début du projet et a nous donné de conseil sur l'utilisation de la plate-forme pendant le même.

2. B.1 Création du site de partage PISAMO / PISA pour MobiMart

Le contrat prévoit la création d'un site web avec la description du projet MobiMart et où publier les différents livrables du projet.

Pour cela, Drupal a été choisi en tant que gestionnaire de contenu à utiliser, qui est une plateforme logicielle open source et il permet la création et la distribution de sites Web dynamiques complexes. Drupal a été choisi donc pour la grande possibilité de personnalisation grâce à sa structure basée sur des modules et des thèmes. La sécurité est également garantie grâce à l'importante communauté Drupal et à la publication de nombreuses mises à jour et corrections de bugs.

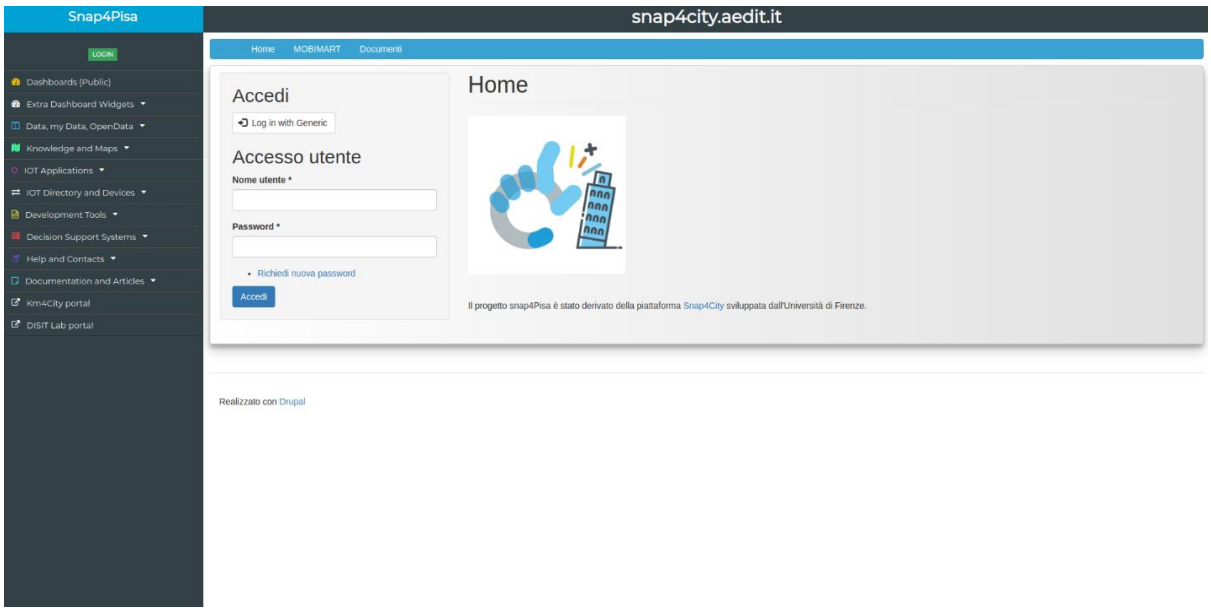
Une instance de Drupal est aussi disponible sur http://snap4city.aedit.it/drupal/

Pour le moment ils ont été installés les modules suivants :

- Ckeditor (permet d'éditer les contenus grâce à une interface “stile Word”);
- OpenId Connect (permet l'intégration avec Keycloak pour l'authentification).

Une configuration de l'instance Drupal et des modules installés a été réalisée afin que l'intégration avec le système d'authentification de la plateforme snap4city soit possible.

Nous avons créé un thème simple basé sur le bootstrap, qui pourra être configuré ultérieurement sur la base des indications qui nous seront fournies. Les pages de description du projet et une section ont été créés pour collecter les documents qui seront produits.



Deux référentiels de projets ont été créés sur GITHUB à l'adresse suivante https://github.com/snap4Pisa:

- **Snap4city-docker** c'est un fork du référentiel officiel de l'Université de Florence sur lequel les personnalisations de la plateforme sont effectuées;
- **Snap4pisa_ingestion** est un référentiel dans lequel toutes les procédures d'ingestion de données qui seront développées dans l'iotApp seront publiées.

3. B.2 Chargement des données

Identification des sources de données possibles

Dans une première phase, nous avons identifié les sources de données possibles que nous avons organisées dans un fichier Excel disponible dans Google Drive à l'adresse suivante (disponible en lecture seule) :
https://docs.google.com/spreadsheets/d/12QpWf8eV6t34C1FkuIuwHnIdqf_gWNaTvNf-Re24xiU/edit#gid=696682826?usp=sharing

16 sources de données ont été identifiées concernant le territoire de la commune de Pise, la personne à contacter et d'autres informations utiles. Un mail a ensuite été envoyé à chaque contact, leur expliquant brièvement le projet et les invitant à une réunion pour comprendre l'intérêt de fournir un type de données, les méthodes, la fréquence de mise à jour ainsi que la structure des données à importer dans le Plate-forme.
Le tableau récapitulatif suivant présente les sources de données identifiées.

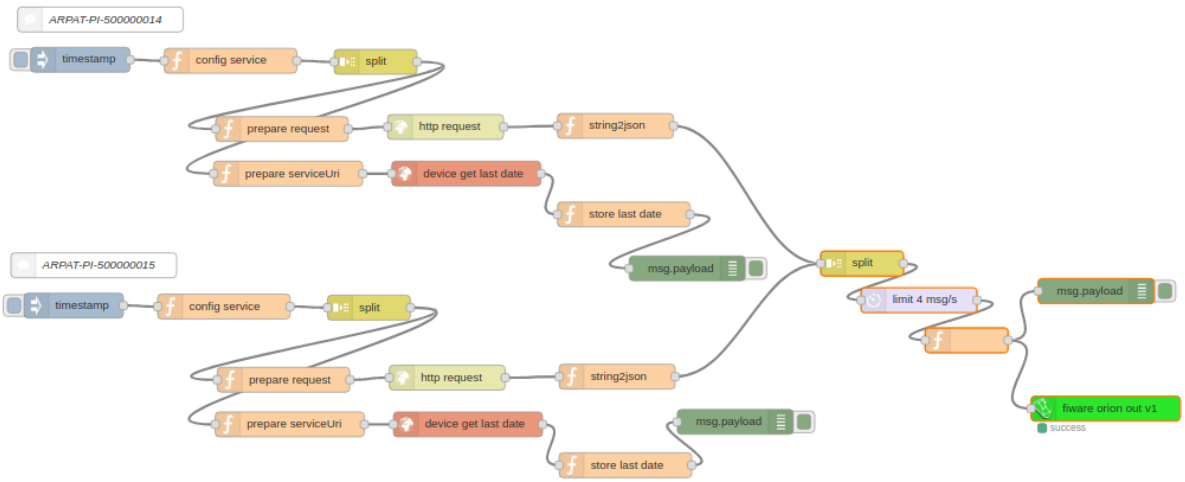
Données gérées	**Contacter**	**état**
People Mover	Igor.Terzariol@leitner.com	In attesa
Bike sharing	g.pin@bicincitta.com	Ricevuto il webservice
Varchi ZTL	sandro.tolaini@autostrade.it	Ricevuto il webservice (blocco firewall)
Flussi di traffico dai sensori della Regione Toscana	walter.pratesi@regione.toscana.it	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

Comme le montre le tableau, tous les fournisseurs n'ont pas répondu aux e-mails envoyés au cours des mois d'août et de septembre.

Développement de procédures d'importation et de chargement de données

Au cours de la période considérée, nous avons commencé à nous familiariser avec la technologie Node-RED utilisée par la plateforme d'importation de données. Node-RED est un outil créé pour la gestion du monde IoT à travers le paradigme des flux de données. La programmation d'une iotApp se fait par blocs; les blocs peuvent contenir des actions automatiques ou personnalisables en écrivant du code javascript simple. Oui, les procédures génériques d'importation des différents types actuellement disponibles ont été écrites et peuvent être insérées ultérieurement sur notre plateforme. Les procédures (sous la forme d'un screeenshot) d'ingestion de données développées sont les suivantes.

Données Arpat



Les données sont chargées automatiquement en définissant une injection répétée en fonction du type de données pour chaque intervalle de temps. Les différentes procédures qui seront mises en œuvre seront publiées dans le repository snap4pisa_ingestion.

En ce qui concerne la documentation, une section spéciale a été créée au sein du site Drupal où tous les documents qui seront produits au cours du projet seront progressivement téléchargés.

Nous avons créé quelques petits guides qui peuvent être modifiés dans le futur si nécessaire. Les guides couvrent les sujets suivants:

- Guide Nodered et iotApp;
- Guide de création de capteurs;
- Guide de création d'un tableau de bord;
- Guide d'installation du système.

Projet spécifique de Deploy à Snap4Pisa

La machine virtuelle qui hébergerait l'ensemble de la plate-forme snap4city a été conçue en accord avec l'Université de Florence. Nous avons commencé par définir les besoins hardware et software.

La plateforme est constituée d'une série de dockers connectés les uns aux autres et qui constituent les services disponibles.

Docker est un projet open source qui utilise les capacités d'isolation des ressources du noyau Linux pour permettre à des «conteneurs» indépendants de coexister sur la même instance Linux, évitant ainsi l'installation et la maintenance d'une machine virtuelle. Un conteneur Docker, contrairement à une machine virtuelle, n'inclut pas de système d'exploitation distinct. Au lieu de cela, il utilise les capacités du noyau et tire parti de l'isolation des ressources pour isoler ce que l'application peut voir du système d'exploitation.

Pour faire coexister une série de conteneurs dans une machine virtuelle, nous avons créé cette dernière avec 100 GB d'espace disque et 32 GB de mémoire. Ces caractéristiques doivent être considérées comme minimales et peuvent être augmentées (manuellement) si nécessaire. Après la création de la machine virtuelle, avec l'aide du support UniFI, nous avons identifié la version de la plateforme qui serait installée et configurée.

La configuration a été choisie DataCity small qui vous permet de gérer une réalité telle que la municipalité de Pise.

Les services activés sont les suivants:

- Dashboard (service principal de toute la plate-forme qui fournit des réponses sous forme de graphiques et de cartes)
- Dashboard-backend (service de développement et de personnalisation de Dashboard);
- Dashboard-cron (service d'automatisation des fonctions comme l'importation de données au fil du temps);
- Personal data;
- Wsserver;
- Synoptics;
- Iotapp-nr1 (service de Node-RED pour le développement de procédures di data ingestion);
- Iotapp-nr2 (service de Node-RED pour le développement de procédures data ingestion);
- Iotapp-nr3 (service de Node-RED pour le développement de procédures data ingestion);
- Zookeeper;
- Kafka;
- Orionbrokerfilter (service qui permet la connexion avec les différentes unités de stockage présentes au sein de la plateforme) ;
- Orion;
- Servicemap (service tomcat);
- Virtuoso-kb (service de storage delle “triplette” pour la définition des points d'intérêt);
- Nifi;
- Elasticsearch (service de storage);
- Kibana (service de storage);
- Dashboarddb (service de storage);
- Ldap-server;

- Keycloak (service d'authentification des utilisateurs);
- Drupal (pour la gestion de contenu).

Une fois que nous avons terminé la première installation de tous les services, nous avons testé le fonctionnement de tous les composants du système dans un domaine interne. Par la suite, toutes les variables d'environnement à modifier pour faire fonctionner le site sur un domaine externe ont été identifiées. Un file di scripting a été créé qui modifie toutes les variables afin de simplifier un éventuel changement futur tel que le changement de domaine.

En étroite collaboration avec le support, nous avons obtenu l'approbation technique des méthodes d'installation et de configuration de l'ensemble de l'environnement.

Durant cette période, nous avons également commencé à optimiser la plateforme en activant, par exemple, le https et lancé la migration des scripts testés sur la plateforme en fonctionnement à l'Université de Florence (https://www.snap4city.org/).

4. B3 Procédures de transformation et de traitement des données

Une série de scripts open source ont été développés avec Node-RED qui permettent l'extraction des données, la transformation de celles-ci dans le format spécifique utilisé par la plate-forme et enfin le chargement des données au sein de la plate-forme Snap4Pisa.

Les données actuellement présentes sur la plateforme sont de deux macro-types:

- Données statiques
- Données en continu dans realtime.

I **Données statiques** sont les points d'intérêt (PoI) présents dans la région de Pise divisés par type; on y trouve par exemple: des bars, des restaurants, des parkings, des guichets automatiques, des hôpitaux, etc.

I **Données en continu** en realtime, sont des données provenant de systèmes Internet of Thing (IoT). Parmi ceux-ci actuellement, les données suivantes ont été importées

- Capteurs de pollution (Arpat)
- Stalles BiciInCittà
- Capteurs de trafic - Region Toscana
- Capteurs ZTL Commune di Pisa

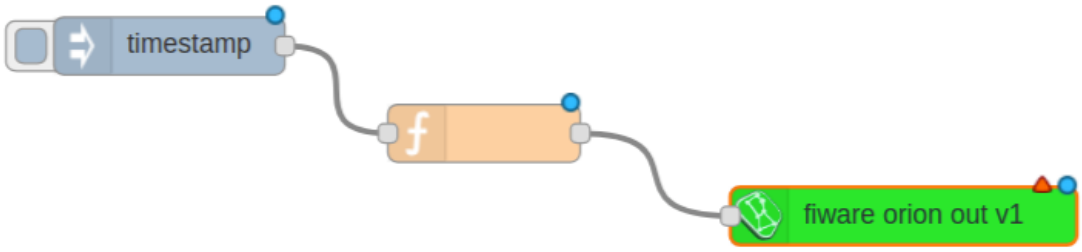
Les procédures d'ingestion de données qui extraient, transforment et chargent les données sont, comme mentionné, développées à l'aide de la technologie Node-RED qui permet la programmation par blocs, où chaque bloc peut être personnalisé en fonction des besoins du cas en question.

À titre d'exemple, pour entrer une donnée, vous avez besoin d'au moins trois blocs comme indiqué dans l'image ci-dessous:

Inject - bloc qui vous permet de démarrer la procédure; il peut également être chronométré pour exécuter automatiquement chaque "unité de temps" que nous définissons comme atomique pour ce type de données;

Funzione - bloc qui vous permet d'écrire du code Javascript et de préparer une donnée qui sera chargée dans la base de données;

Fiware orion out v1 - bloc qui vous permet d'insérer des données dans contextBroker o database.



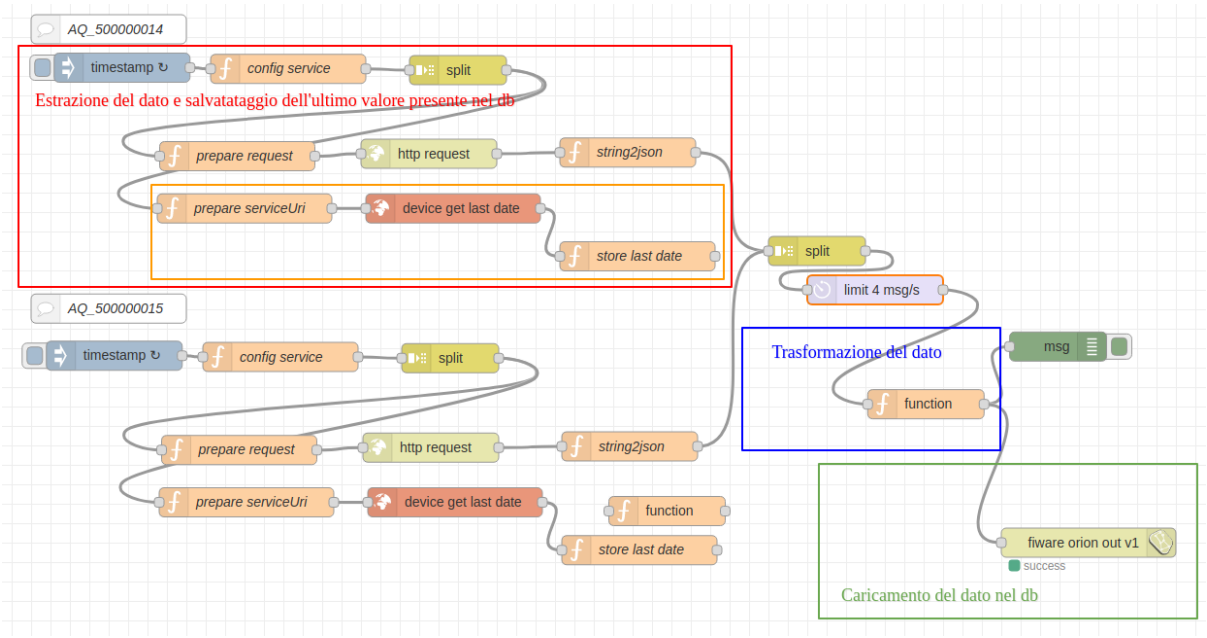
Dans le bloc de la fonction peut être inséré un code Javascript qui définit le format avec lequel les données seront stockées dans la base de données. Par exemple, ce code court crée une nouvelle mesure détectée par le capteur "sensor_weather" avec les valeurs de température et d'humidité.

```
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;
```

Les données chargées dans la plateforme selon cette structure peuvent ensuite être envoyées vers les tableaux de bord où elles seront affichées dans les différentes formes définies par les widgets disponibles; cet aspect du système sera décrit dans la section B4 plus loin dans ce rapport.

B3.1 Import donnes Arpat

Deux flux ont été créés, un pour chaque capteur présent dans la commune de Pise, avec une dernière partie en commun qui traite du chargement final dans la base de données. Voici le script Node-RED qui effectue l'extraction en rouge, la transformation en bleu via un script Javascript et enfin le chargement des données en vert:



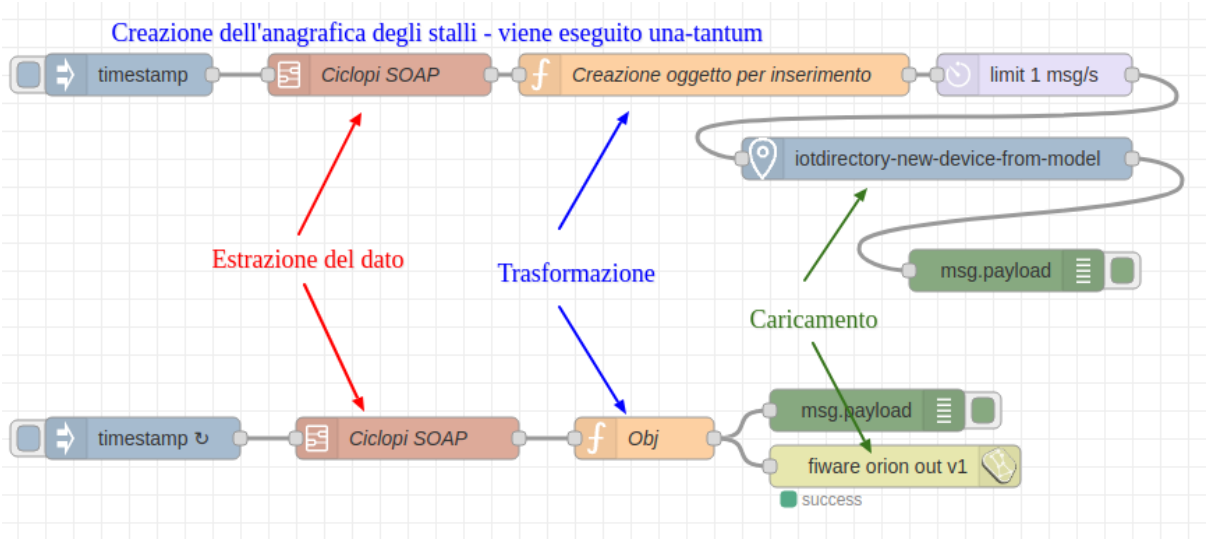
Le script, afin de limiter la duplication des données, mémorise la dernière valeur saisie (sous-système orange) afin de n'entrer que les nouvelles valeurs détectées par le capteur.

Dato	Sensori Arpat
Step del dato	Horaire
Ingest	À l'heure

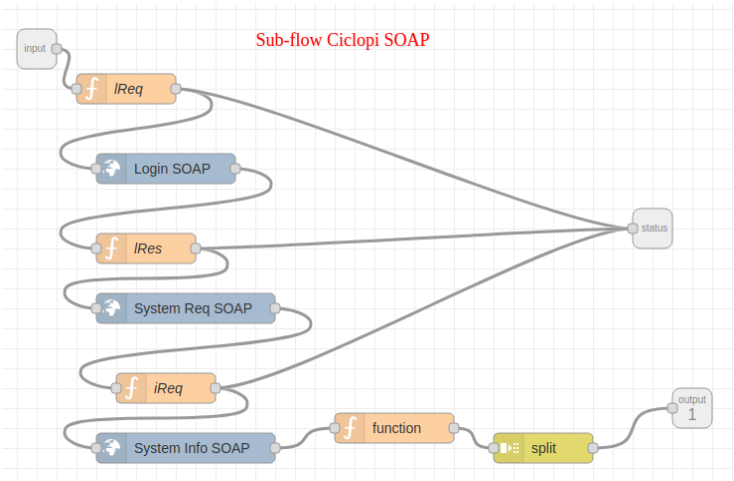
B3.2 Stalles BiciInCittà

Deux flux ont été créés, l'un pour saisir les données personnelles de chaque stand et l'autre pour saisir les données sur la disponibilité des vélos à l'intérieur du stand.

Voici le script Node-RED qui effectue l'extraction en rouge, la transformation en bleu via un script javascript et enfin le chargement des données en vert:



Dans ce cas, un sous-flux a été créé qui effectue une série d'opérations dans le but spécifique d'extraire les données. L'image avec les blocs qui traitent de l'extraction de données sont montrées dans l'image suivante:

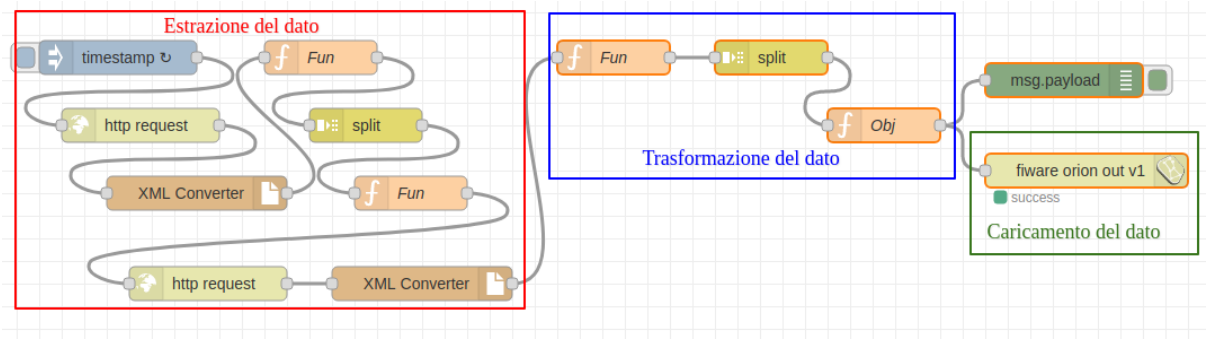


Donné	Capteurs BikeSharing
Étape des données	Real time
Ingest	Toutes les 5 minutes

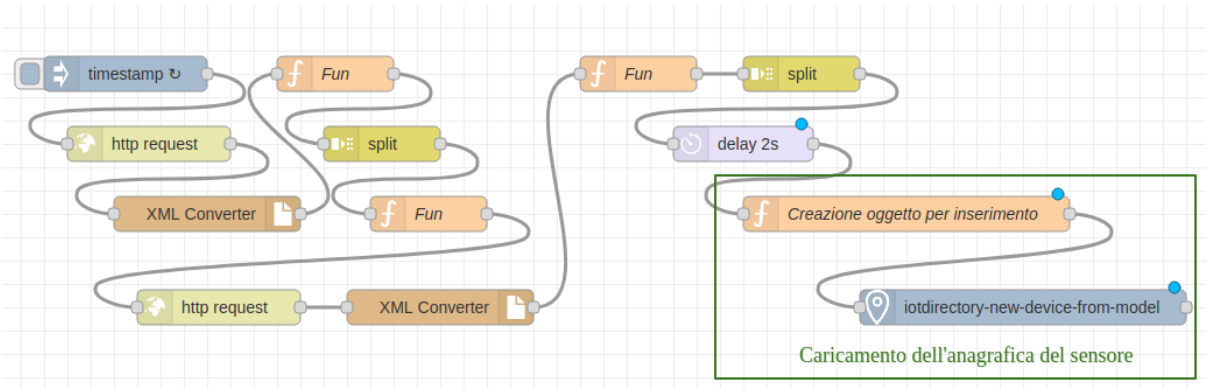
B3.3 Capteurs de trafic

Dans ce cas également, deux flux ont été créés : l'un pour saisir les données personnelles de chaque capteur de trafic et l'autre pour saisir les données de passage des véhicules.

Ci-dessous se trouve le script Node-RED qui effectue l'extraction en rouge, la transformation en bleu via un script Javascript et enfin le chargement des données en vert:



Le deuxième flux est utilisé de manière ponctuelle pour entrer dans le registre des différents capteurs de trafic installés sur le territoire de la commune de Pise.



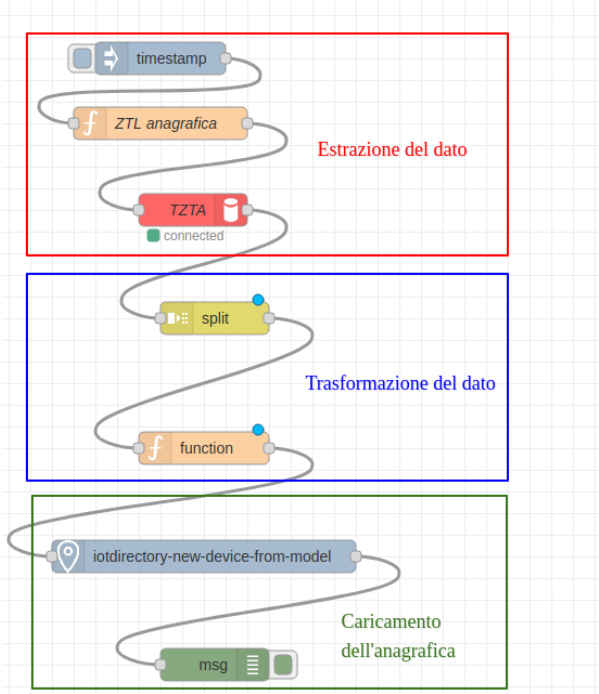
Dato	Capteurs de flux de trafic
Step del dato	Real time
Ingest	Toutes les 3 minutes

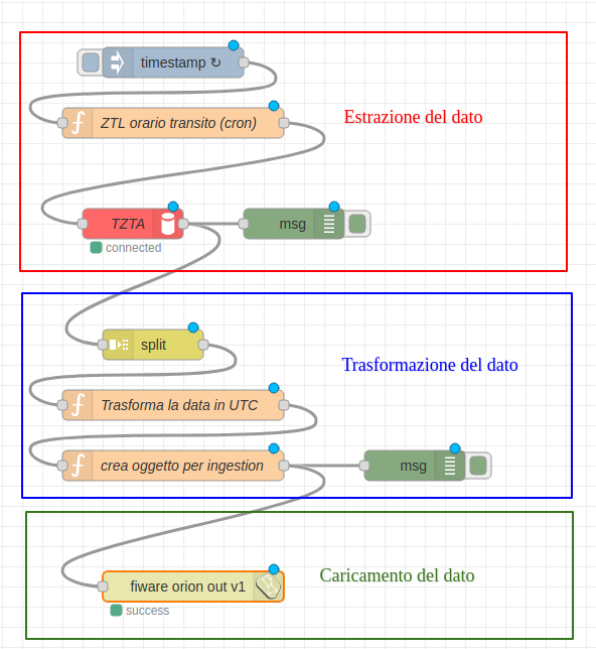
B3.4 Capteurs ZTL Commune de Pisa

Dans ce cas également, deux flux ont été créés, l'un pour saisir les données personnelles de chaque capteur ZTL et l'autre pour saisir les données d'enquête véhicule.

Les deux scripts Node-RED suivent: le premier effectue l'extraction en rouge, la transformation en bleu via un script Javascript et enfin le chargement des données

vertes relatives au registre de la caméra ZTL tandis que le second insère les données du passage dans temps réel.





Donné	Capteurs ZTL
Étape des données	Real time
Ingest	Toutes les 2 minutes

B3.5 Parking

Dans ce cas un script externe a été créé, développé en python. Après avoir configuré toutes les caméras pour envoyer les photos des enquêtes, une application python Flash a été créée qui interroge toutes les images afin d'extrapoler les données d'intérêt. Ci-dessous le 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)
```

Enfin un cron a été configuré pour s'exécuter toutes les minutes.

Dato	Parcheggi
Step del dato	Real time
Ingest	Ogni 1 minuto

5. B4 tableaux de bord et dashboard

Dans cette phase du projet, nous avons commencé à développer les premiers tableaux de bord pour la visualisation des données. Les tableaux de bord sont un outil conçu pour afficher des données via des widgets pré-emballés.

Les dashboard actuellement présents dans le système constituent une première approche de l'outil visant à comprendre son potentiel en attendant de créer des personnalisations de celui-ci sur la base des retours d'expérience recueillis lors de la réunion de co-création et de co-conception qui se tient dans ces jours avec les parties prenantes invités à y assister.

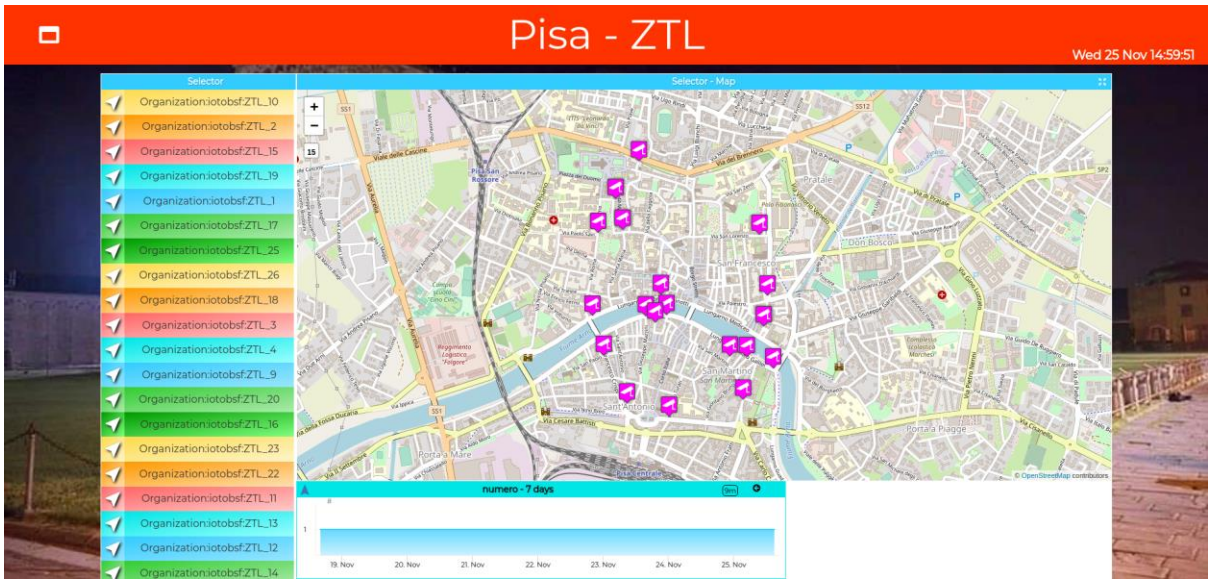
B4.1 Dashboard polluants

Le tableau de bord ci-dessous montre la tendance de la qualité de l'air détectée par les deux unités de contrôle présentes dans la municipalité de Pise. Pour chaque paramètre NO2, O3 et CO, il existe trois widgets:

- Comparaison quotidienne
- Comparaison hebdomadaire
- Dernière valeur détectée

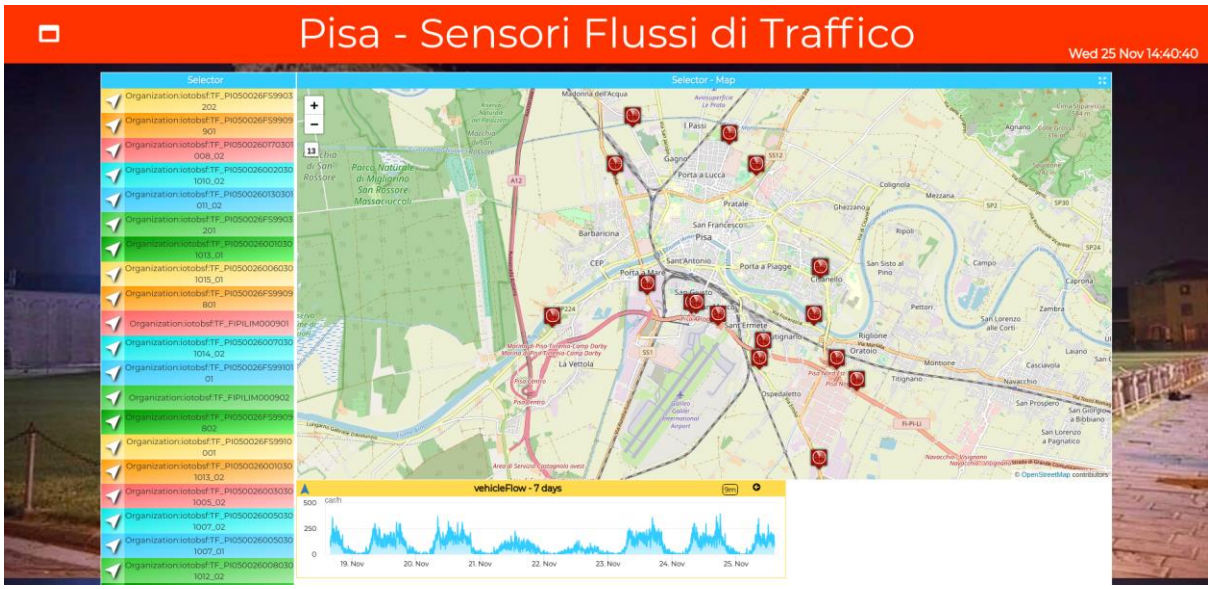
B4.2 Dashboard ZTL

Le dashboard ci-dessous montre la position des caméras ZTL présentes sur le territoire de la municipalité de Pise.



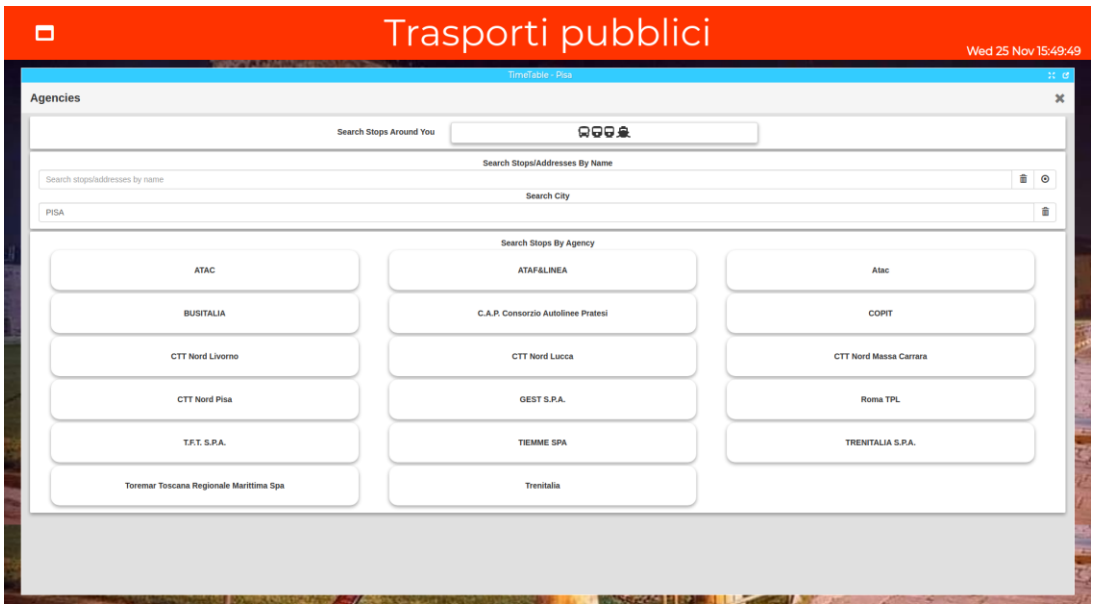
B4.3 Dashboard Flux de trafic

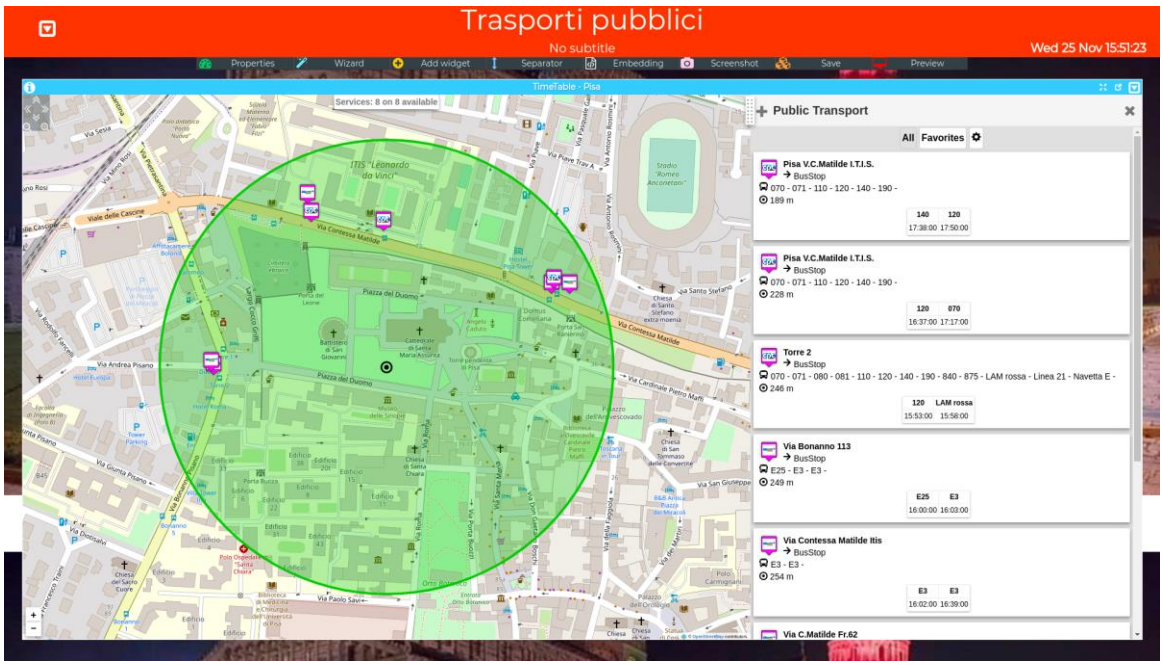
Le dashboard ci-dessous montre la position des capteurs de détection de flux de trafic présents sur le territoire de la municipalité de Pise.



B4.4 Transport public

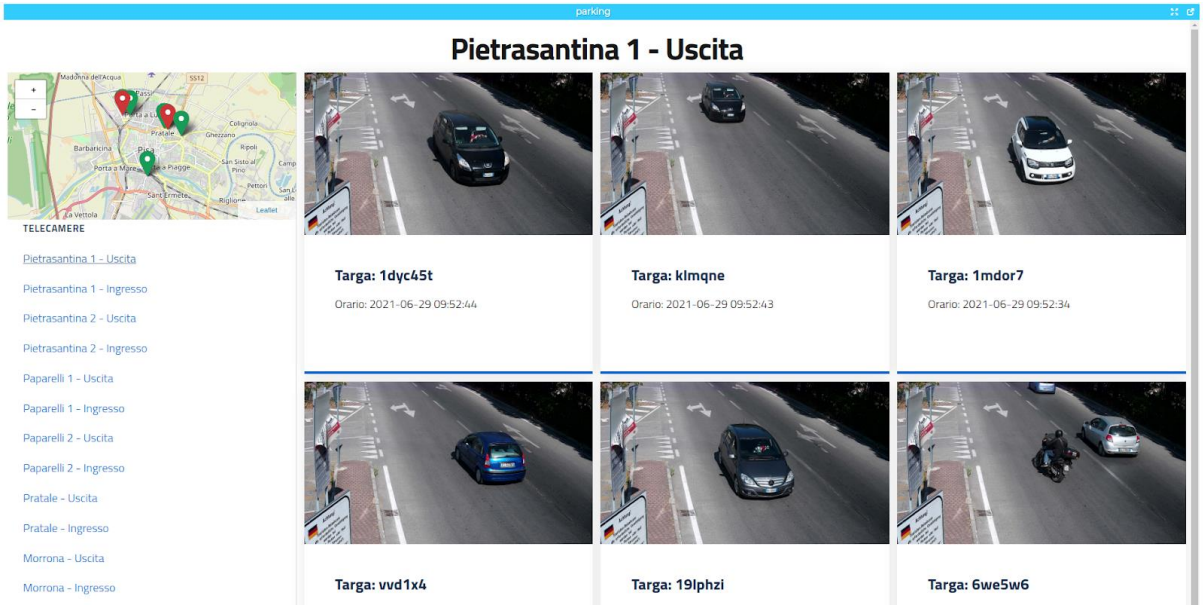
Le dashboard ci-dessous montre les transports en commun. Vous pouvez rechercher un service en fonction de l'agence qui fournit le service (il existe des agences opérant dans toute la Toscane), ou vous pouvez rechercher l'arrêt le plus proche grâce à la position GPS de l'appareil. Les deux images ci-dessous montrent ces deux types.



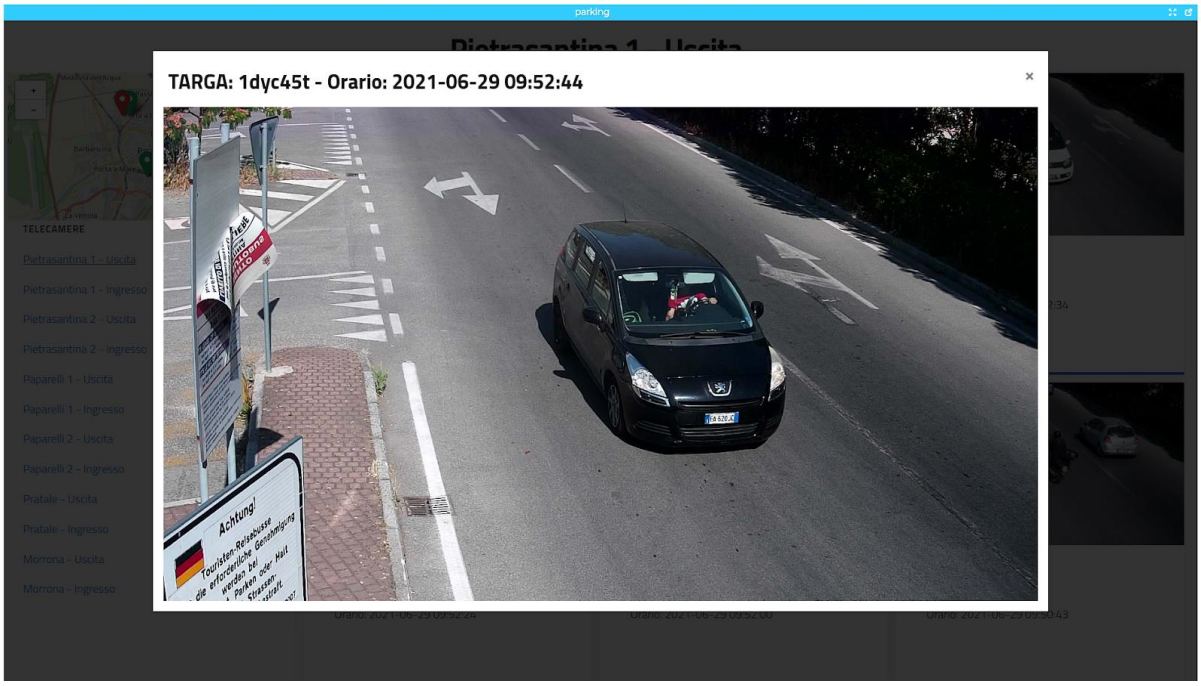


B4.5 Parking

Le tableau de bord ci-dessous présente les accès aux parkings qui ont été installés et configurés au cours du projet.



Le tableau de bord affiche une carte avec le positionnement des caméras dans les parkings. Les pointeurs sont représentés en deux couleurs pour identifier les caméras entrant dans le parking depuis la sortie. Il existe également une série de liens vers les caméras; en cliquant sur chacun d'eux, vous accédez à la caméra correspondante. La partie droite du tableau de bord affiche les 12 dernières mesures prises. Chaque enquête montre la photographie du véhicule avec la plaque d'immatriculation et l'heure d'entrée / sortie. En cliquant sur une enquête, le système affichera le zoom comme indiqué dans l'image suivante.



6. B5 Création de l’App dans a SNAP

En février, nous avons commencé à travailler sur une version prototype de l'application à partir d'un kit de démarrage mis à disposition par l'Université de Florence via la repository https://github.com/disit/snap4cityAppKit

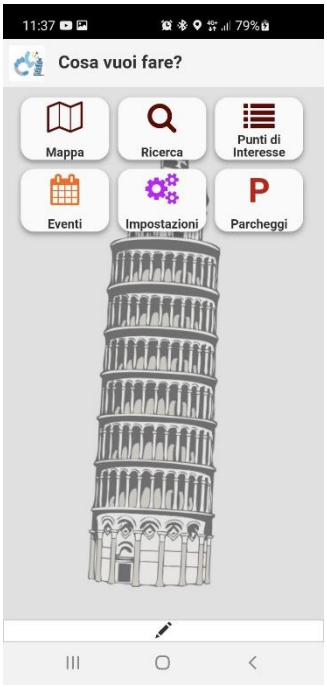
L'application est développée avec le framework cordova qui permet d'utiliser le même code source également pour la génération d'une version pour l'environnement Apple.

Actuellement, l'application n'a pas été chargée sur la boutique Android mais est disponible sur le lien suivant:

https://snap4city.aedit.it/drupal//sites/default/files/snap.apk

Voilà quelque screenshot dell’app.

Splash screen dell'app



Menu principal de l'application

Exemple de recherche de services proches de mon poste.

Exemple de recherche de services en fonction du type demandé.

L'application peut être développée ultérieurement en ajoutant de nouveaux boutons dans le menu principal.

7. B7 Cours de formation et de recyclage

Pendant toute la durée du projet, nous avons été en contact avec l'Université de Florence et en particulier avec le prof. Nesi, responsable de la plateforme snap4city qui, avec l'aide de ses collaborateurs, nous a accompagnés dans la plupart de nos demandes.

Afin de minimiser le temps d'intervention, nous nous sommes mis d'accord sur la création d'un groupe de travail sur la plateforme Skype. Le chat toujours actif a donné à chacun la connaissance des sujets de discussion et l'opportunité d'intervenir.

Au début du projet, des sessions d'introduction à la plateforme ont été organisées au cours desquelles l'équipe Aedit a pu organiser de petites tâches de travail afin de mettre en pratique ce qui était expliqué. À partir de la mise en œuvre, des besoins particuliers pourraient survenir, qui ont été traités soit par des questions / réponses sur le chat de groupe, soit par un appel destiné à analyser et résoudre le problème spécifique.

8. B8 Formation

Pendant toute la durée du projet, des guides ont été créés vers la plateforme qui permettent, avec les tutoriels rédigés par l'Université de Florence, d'être autonome.

Les quatre guides créés et disponibles sur ce lien traitent des sujets suivants:

- Guide de Nodered et iotApps
- Guide de création des capteurs
- Guide de création d'un dashboard

  Guide d'installation du système

Les didacticiels peuvent être consultés sur le lien suivant et couvrent les sujets suivants:

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

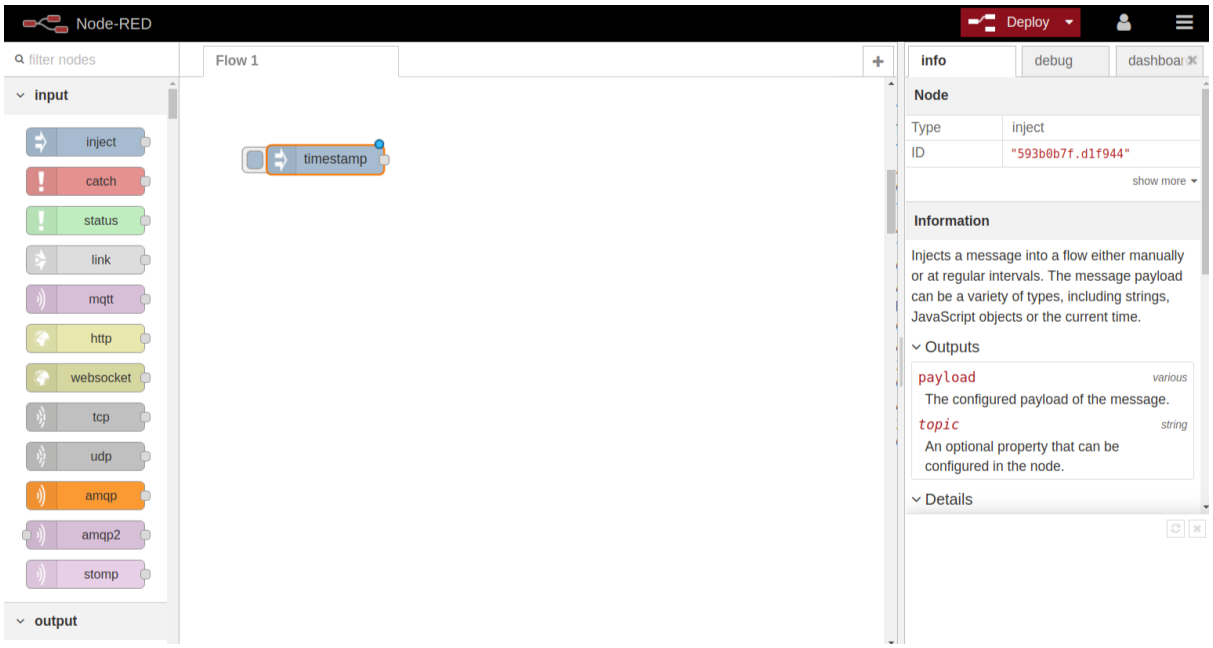
- Data Ingestion processes
- System and Deploy Install
- Smart City API: Web & Mob. App

ANNEX

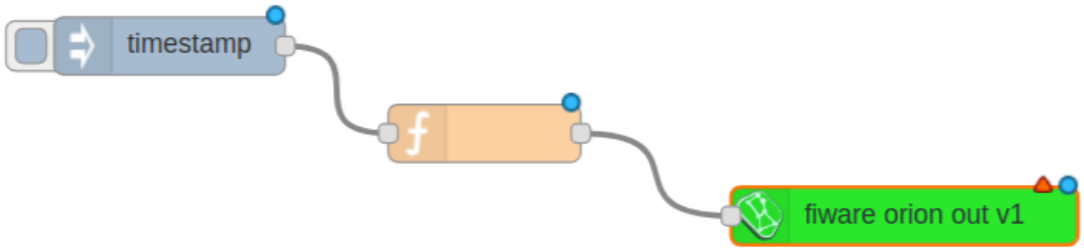
Guide de base sur Nodered et iotApps

Saisir des données IoT avec un IoTApp

- accéder aux applications IOT et créer une nouvelle IOTApp; le type d'application doit être basique. Il est également possible de réutiliser une application IoT que vous avez déjà créée; il est conseillé d'insérer plus d'une procédure dans 1 iotApp pour pouvoir les gérer de manière plus simple et ne pas occuper trop de ressources système
- iotAPP a des blocs dans la colonne de gauche; sélectionner le block “inject” et faire-le glisser dans le board; Le bloc est utilisé pour avoir une commande pour lancer une série d'opérations en cascade; après l'avoir fait glisser, cliquer sur “deploy” pour "publier" l'application. A ce stade, si vous cliquez sur le "bouton" à gauche du bloc, le système "injecte" (insère) une variable dans le système, dans ce cas il insère la date actuelle (horodatage) (apparaît dans le message " Injection réussie: horodatage) "



- au moins trois blocs sont nécessaires pour pouvoir saisir des données; insérer les blocs et connecter-les ensemble comme indiqué dans l'image suivante:
  - inject: déjà vu pour lancer la procédure
  - function: est un bloc qui vous permet d'ajouter du code javascript
  - fiware_orion_out_v1: ce bloc permet de saisir des données dans un contexte de type “Orion Context Broker” en version v1; c'est l'appareil actuellement le plus populaire, dans la bibliothèque il y a des blocs dédiés à contextBroker d'autres versions

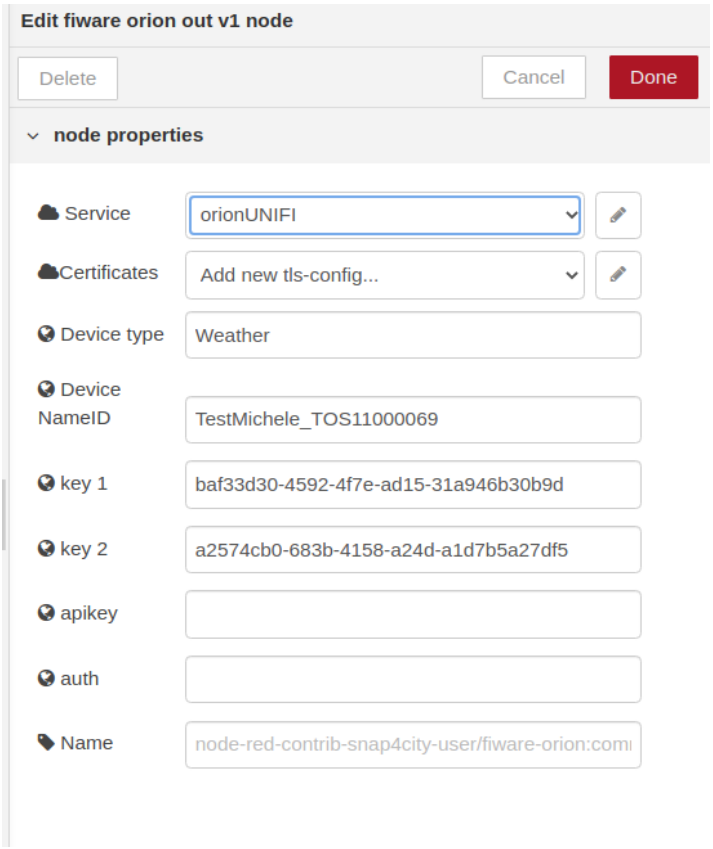


- cliquer sur le bloc fonction (celui du centre) et écriver la fonction de saisie suivante; dans l'exemple que nous montrons, nous insérons une donnée avec la date actuelle et des valeurs fixes à chaque fois que vous cliquez sur injecter
  - id: le code exact de l'appareil que vous avez créé
  - type: le type que vous avez défini dans iotModel et iotDevice
  - name: les noms exacts des attributs définis dans le iotModel
  - value: les valeurs que nous voulons insérer; new Date () insère la date actuelle

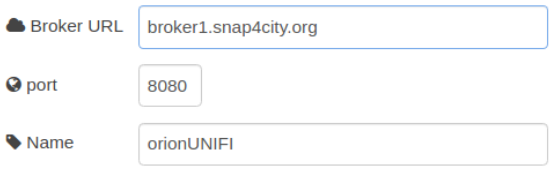
```
var data={
 "id": "test_diego_pisa",
 "type": "misura",
 "attributes":[
   {"name": "dateObserved", "value": new Date().toISOString(), "type": "time"},
   {"name": "temperature", "value": 32.1 , "type": "float"},
   {"name": "humidity", "value": 44.1 , "type": "float"}
 ]
}
var msg={payload: data}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

Attention : le champ d'heure (dateOserved) doit être au format texte ISO: es: “2020-08-07T12:51:50.548Z”; IN JS à partir d'un champ de données, vous obtenez la chaîne ISO avec la fonction toISOString ();

- Configurer il Fiware orion out v1
  - ou double click
  - ou insérer orionUNIFI
  - ou insérer k1 e k2 (si vous ne les avez pas, vous pouvez les trouver dans le device)



Où le service doit être modifié au crayon en insérant **orionUNIFI** comme dans la figure suivante



Important: Il est possible de passer l'authentification du capteur dans le message qui est envoyé à Fiware orion out v1; pour ce faire, préparez simplement l'objet comme suit:

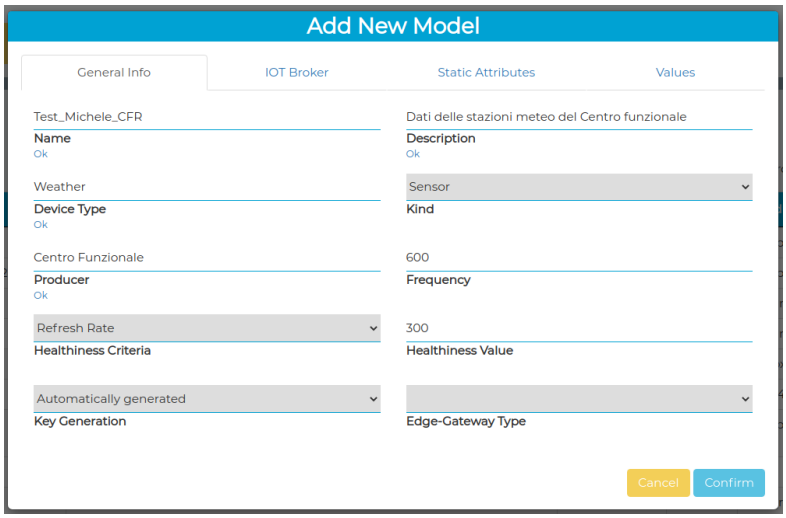
```
var k1 = “f6efd060-375b-4f40-af09-41e0814f7039”;
var k2 = “108bfe91-8140-41fd-92d8-c2b317c3cef8”;
...
var auth={"k1": k1,"k2": k2,"apikey":"apikey","basicAuth": "basicAuthKey"};
return {"auth": auth, "payload":ret}
```

Guide de création des capteurs

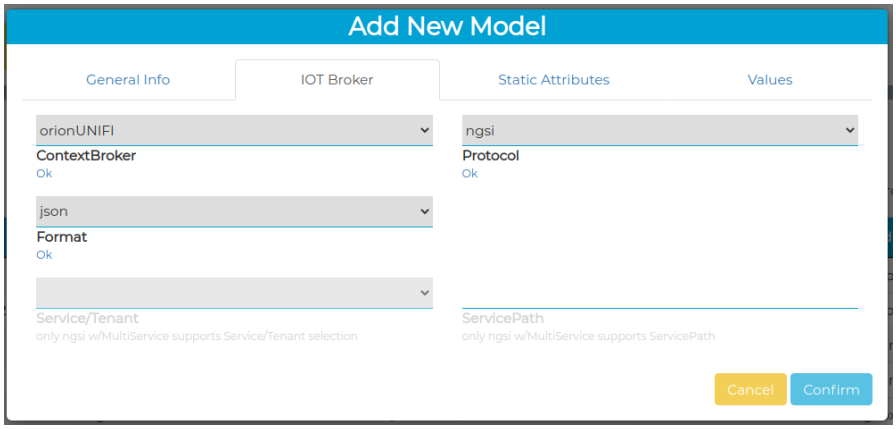
Créer un IoT Device Model

Dans le lien suivant, une description complète en anglais de la façon d'insérer et de configurer un https://www.snap4city.org/drupal/node/591)

- le nom, la description et le type de l'appareil doivent être définis; dans ce cas, les données d'une station météo sont saisies, puis la météo est utilisée comme type



- cliquer sur “IOT Broker” et choisissez le courtier dans la liste déroulante

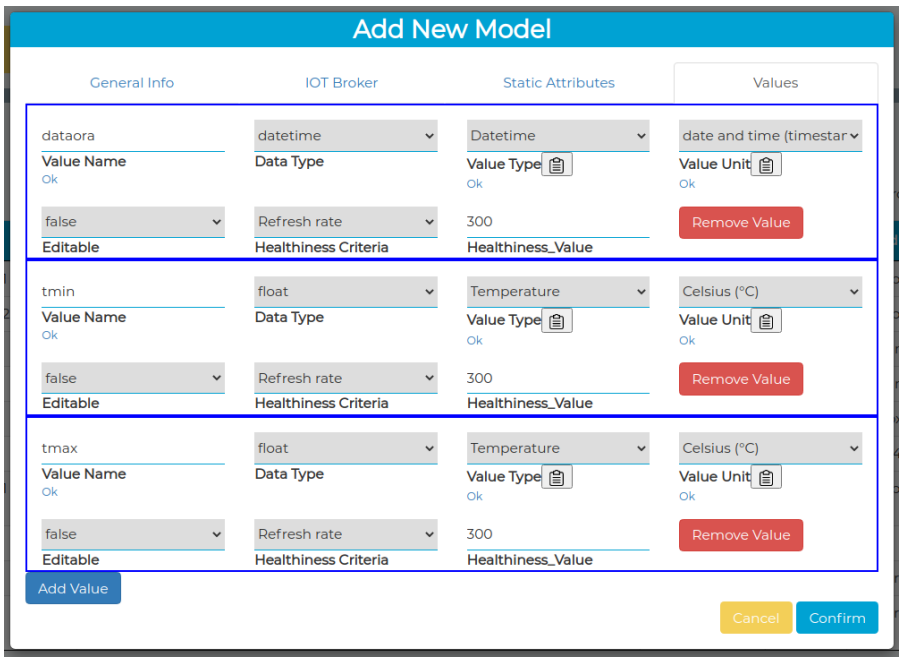


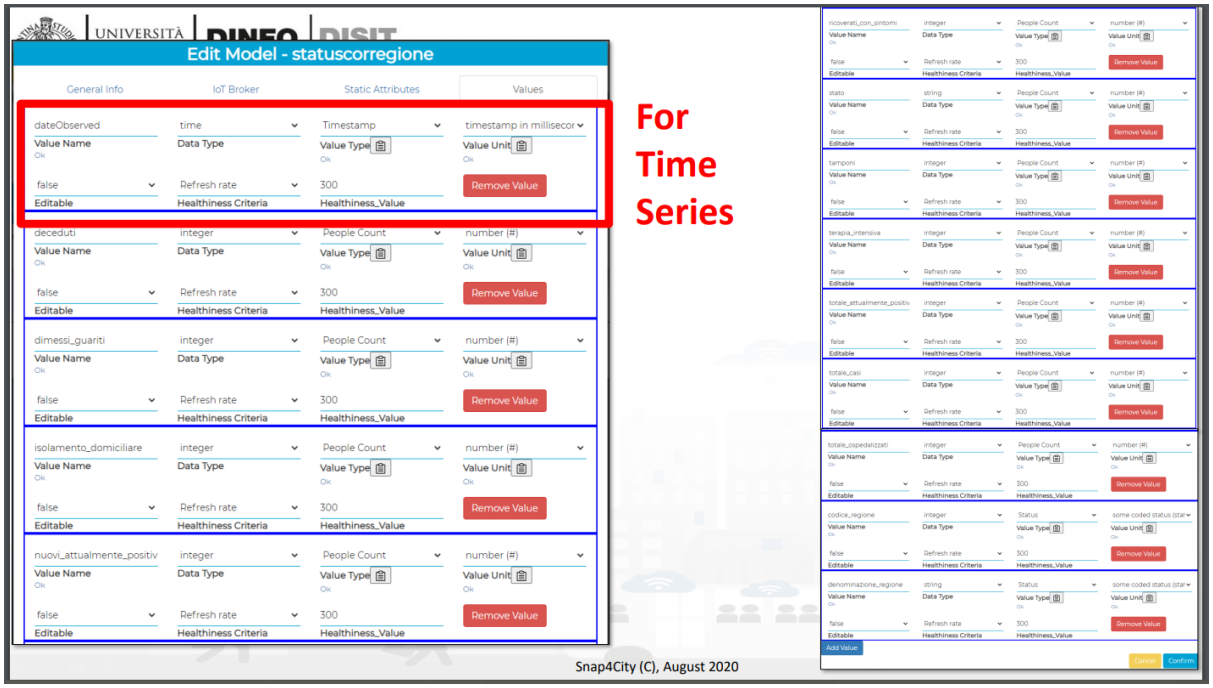
dans tab Values vous devez définir la structure des données qui seront acquises par le modèle
pour chaque variable ajouter une “ligne” avec Add Value
sélectionner la data type (date, chaîne, float, ...)
sélectionner le value type
sélectionner le value unitbnn0

Attention!!!. Dans le dataModel si vous avez affaire à une série chronologique, vous devez créer un champ codé de cette manière:

- le nom doit être dateObserved (vérifier)
- le Data Type ce doit être Time
- le Value Type ce doit être **Timestamp**
- le valued ce doit être timestamp en milliseconde

S'il n'est pas bien codé, le système ne reconnaît pas la date de mesure.

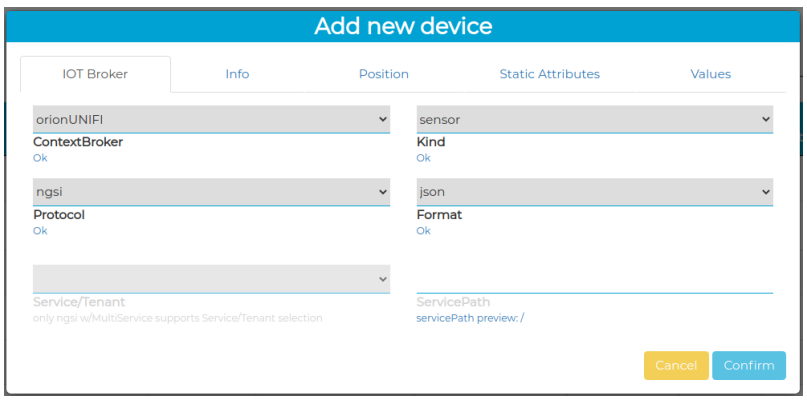




Enregistrement manuel d'un IoTDevice

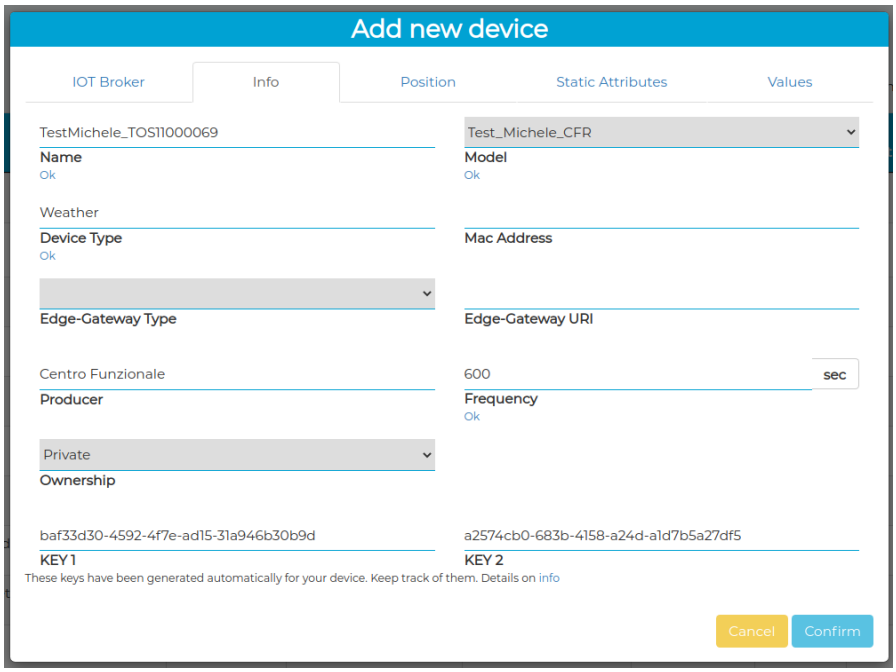
Ajouter un nouveau IoTDevice

- sélectionner le contextBroker

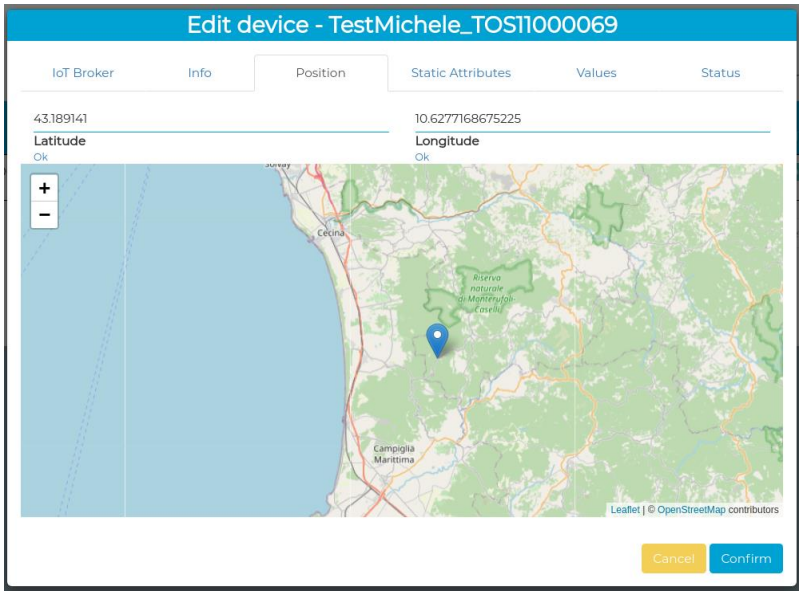


- Dans tab Info:
  - attribuer un nom

- ou sélectionner le modèle (créé au point précédent)
- ou enregistrer les clés séparément KEY1 est KEY2



- Dans tab Position entrer les coordonnées



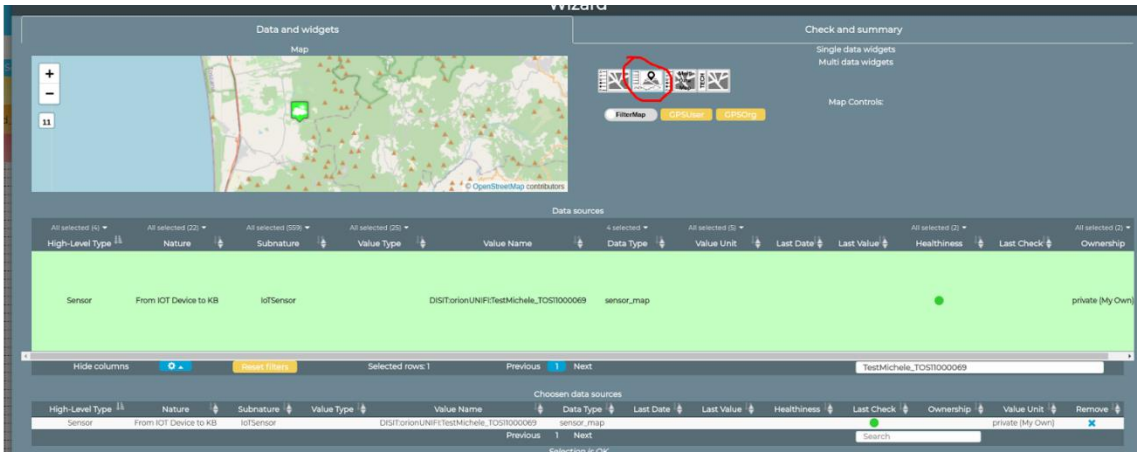
Enregistrer device créé

Suggestion: Dans le cas de la définition de plusieurs capteurs du même type (en général on pourrait s'étendre à tous les capteurs du projet Pisa), on peut paramétrer les mêmes k1 et k2 pour tous les capteurs.

Guide de création Dashboard

Créer un dashboard / ajouter le widget à un dashboard existant

- Ouvrer le Wizard
- Rechercher le nom de device qui a été créé
- Sélectionner la première ligne (celle avec le point vert)
- Sélectionner le type de widget mis en évidence dans la figure ci-dessous.



Guide d'installation du système

Installation de snap4city

Préparer la voiture

- La RAM doit être au moins 32 GB (sinon les machines peuvent tomber en panne)
- Pour elasticsearch vous devez configurer le max_map_count de la voiture host;
- s'il n'a pas cette valeur, il pourrait avoir des problèmes de mémoire (le container docker est immédiatement arrêté)
    - nano /etc/sysctl.conf
    - # ajouter la ligne suivante
    - vm.max_map_count=262144
    - sysctl -p

Télécharger le docker-composer et installez-le en suivant les instructions

```
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 des machines et post setup

Attention!!!! Il est nécessaire de vérifier que toutes les machines ont démarré. Plus précisément, les machines avec JAVA peuvent poser des problèmes de RAM et planter. Aussi ElasticSearch

cela peut avoir des problèmes si le serveur n'est pas bien configuré.

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

À ce stade, suivez les instructions et cela devrait fonctionner dans localhost..

Installation dans un environnement d'exploitation

Afin de l'installer en fonctionnement, il est nécessaire d'attribuer un domaine à la machine puis de changer le nom du domaine lui-même dans tous les scripts. Références aux menus dans le database mySQL. Nous avons fait un script **snap4city_onpremise_setup.sh** faire un backup des soleils file qui sera modifié avec le nom de domaine que nous donnerons à l'environnement d'exploitation (dans notre cas snap4city.aedit.it)

Après run du script je dois être changé redirectURI dans lequel je suis keycloak.

- keycloack:
  - ou connecter-vous au site avec vos identifiants admin / admin
  - ou aller au clients
  - ou modifier ce qui suit Clients
    - Js-grp-client
    - Js-kpi-client
    - Js-synoptic-client
    - Nodered
    - Php-dashboard-builder
    - Php-iot-directory
    - Php-notificator
    - Php-ownership-api
  - ou pour chaque ajouter un Redirect URIs = <nome_del_dominio>

Modifications à docker-compose

- ajouter un volume à keystore ne pas avoir à perdre à chaque restart la configuration

Dans docker-compose ajoutez le volume:

keycloak:

...

volumes:

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

La configuration de snap4city doit être "enregistrée" dans le volume en cours de création via la commande à exécuter à l'intérieur du 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