Agenzia nazionale per le nuove tecnologie, l'energia e lo sviluppo economico sostenibile

Ministero della Transizione Ecologica

Ricerca di Sistema elettrico

Sviluppo di una piattaforma di gestione dei dati

M. Orsini, L. Magnotta, E. Mescoli, A. Livaldi

DataRiver
open source data management

Report RdS/PTR(2021)/037

Sviluppo di una piattaforma di gestione dei dati

M. Orsini, L. Magnotta, E. Mescoli, A. Livaldi

Giugno 2021

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero della Transizione Ecologica - ENEA
Piano Triennale di Realizzazione 2019-2021 - II annualità
Obiettivo: Tecnologie
Progetto: Tecnologie per la penetrazione efficiente del vettore elettrico negli usi finali
Work package: Local Energy District
Linea di attività: LA1.47 - Progettazione piattaforma LEC, implementazione infrastruttura e servizi orizzontali
Responsabile del Progetto: Claudia Meloni, ENEA
Responsabile del Work package: Claudia Meloni, ENEA

Il presente documento descrive le attività di ricerca svolte all'interno del Contratto "Sviluppo di una piattaforma di gestione dati"
Responsabile Unico del Procedimento ENEA: Gianluca D'Agosta
Responsabile del Contratto per il Contraente DataRiver srl: Mirko Orsini

Indice

- SOMMARIO 4

1. DESCRIZIONE DEL PROGETTO 5

1.1. PIANO DI PROGETTO 5
1.2. ENERGY COMMUNITY DATA PLATFORM 6

2. STACK TECNOLOGICO 7

3. ARCHITETTURA 8

3.1. WRAPPER 9
3.2. INTEGRATION (MOMIS MEDIATOR) 10
3.3. STORAGE 10
3.4. QUERYING E EXPORT 11
3.5. SCHEDULING DI SCRIPT CUSTOMIZZATI 11
3.6. SOURCE MANAGEMENT DASHBOARD 11

4. SVILUPPO MODULI DI ENERGY COMMUNITY DATA PLATFORM 12

4.1. WRAPPER 12
4.1.1. *Dati dell'Area Triage* *12*
- *Flusso EnelX con dati al secondo* *13*
- *Flusso dei dati meteo delle centraline di Scandiano (RE)* *13*
- *Flusso Dati ARERA* *14*

4.1.2. *Dati del Data Collector* *15*
- *Dati gestionali - DB Selfuser* *15*
- *Comunità virtuale uffici e abitazioni: DB Selfuser* *15*
- *Dati letture - DB Selfuser* *16*

4.1.2.1. Weatherdata e Counterreading 16
4.1.2.2. Consumo elettrico palazzine A e B 16

- *Dati relativi al caso d'uso del PELL* *17*

4.2. INTEGRATION 22
- *Caricamento delle sorgenti* *22*
- *Annotazione dei campi* *23*
- *Generazione delle relazioni semantiche, definizione e raffinamento delle regole di mapping* *25*
- *Data Cleaning* *26*

4.3. STORAGE 32
- *Configurazione dei flussi di integrazione* *32*
- *Configurazione del servizio di data retention* *34*
- *Configurazione del servizio di scheduling degli script customizzati* *35*

4.4. QUERYING E EXPORT 36
4.4.1. API Esportazione in UrbanDataset *38*
4.4.2. Struttura Modulo di Esportazione in UrbanDataset *40*
4.4.3. API Query Dinamiche *46*

4.5. SOURCE MANAGEMENT DASHBOARD 47
- *Integration Cycles Details e Retention Cycles Details* *48*
- *Integration Performance e Retention Performance* *48*
- *Script Customizzati* *48*
- *Monitoraggio job di Integrazione, retention ed esecuzione script* *48*

5. STRUTTURA CODICE ECD PLATFORM 49

Sommario

Il presente rapporto tecnico descrive le attività svolte per lo sviluppo della piattaforma Energy Community Data Platform per l'integrazione e la gestione dei dati delle Comunità Energetiche. La piattaforma è stata sviluppata per consentire l'acquisizione dei dati provenienti dai diversi flussi in forma eterogenea, la gestione appropriata dei flussi di raccolta e della memorizzazione dei dati dalle diverse sorgenti e utenze (private e pubbliche), le funzionalità di integrazione, trasformazione e pulizia per predisporre i dati all'interrogazione, all'analisi e alla visualizzazione. Nel documento vengono descritte l'architettura ed i moduli software della piattaforma sviluppati e le tecnologie utilizzate.

1. Descrizione del progetto

La piattaforma middleware per l'integrazione e la gestione dei dati delle Comunità Energetiche Energy Community Data Platform è stata progettata in maniera modulare come un sistema flessibile e aperto, per permettere facilmente future estensioni per l'implementazione di nuove funzionalità, diverse modalità di connessione e raccolta dati da dispositivi o servizi e per prevedere l'integrazione con gli altri componenti dell'architettura software e con sistemi software presenti all'interno della Comunità Energetica.
La piattaforma middleware per l'integrazione e gestione dei dati è stata sviluppata secondo le specifiche e la documentazione tecnica fornita da ENEA e grazie ad una interazione continua tra il team di DataRiver ed il team di progetto di ENEA.
La piattaforma è basata sul sistema Open Source MOMIS (Mediator EnvirOnment for Multiple Information Sources) che adotta un approccio semantico per l'integrazione delle sorgenti dati e facilita l'acquisizione di nuove sorgenti.
La piattaforma middleware Energy Community Data Platform per l'integrazione e gestione dei dati è sviluppata secondo l'architettura Wrapper/Mediator del sistema MOMIS che è in grado di aggregare e rendere omogenee informazioni provenienti da sorgenti dati eterogenee, sia strutturate che semi-strutturate, in modo semi-automatico. I punti di forza della soluzione proposta sono la flessibilità e la scalabilità.

1.1. Piano di progetto

Di seguito è riportato il GANTT di progetto, che descrive le Milestone in cui sono state raggruppate le attività, specificandone durata, relazioni di precedenza e percorso critico:

ATTIVITA'	MESI					
	1	2	3	4	5	6
MILESTONE 1						
MILESTONE 2						
MILESTONE 3						
MILESTONE 4						

Il piano delle attività di progetto ha previsto quattro principali **milestone** (M1, M2, M3, M4) di seguito descritte:

Milestone 1 (**M1**):

- Analisi e Progettazione della piattaforma middleware per l'integrazione dei dati
- Sviluppo wrapper per comunicazione con DeltaLake
- Sviluppo wrapper per comunicazione con PostgreSQL con TimescaleDB
- Sviluppo servizi software per lo storage dei dati
- Creazione prima versione dell'ontologia REC (relativa a Energy Box, Distributore, Cabine Elettriche)
- Creazione prima versione dello schema integrato attraverso modulo mediatore
- Sviluppo servizi per l'interrogazione e l'esportazione dei dati nei formati CSV, UrbanDataSet
- Sviluppo draft Dashboard con primi indicatori

Milestone 2 (**M2**):

- Definizione dei template delle proprietà dei POD (funzioni standard per il calcolo)
- Sviluppo wrapper per configurazione dei POD e applicazione delle funzioni di calcolo
- Sviluppo wrapper per la comunicazione con il DataHall e DataCollector
- Sviluppo funzionalità per l'ingestion dei dati dai POD
- Creazione versione completa dell'ontologia REC
- Creazione versione completa dello schema integrato
- Sviluppo funzionalità del servizio di analisi
- Sviluppo Dashboard con indicatori

Milestone 3 (**M3**):

- Sviluppo servizi per il source management (tipologia, variabili monitorate, funzioni di calcolo)
- Sviluppo Dashboard con indicatori completi

Milestone 4 (**M4**):

- Sviluppo logica gestione ruoli e permessi sui dati
- Test finale e validazione funzionalità della Energy Community Data Platform

1.2. Energy Community Data Platform

La piattaforma Energy Community Data Platform è stata sviluppata secondo l'architettura Wrapper/Mediator del sistema MOMIS ed in particolare la sua estensione specifica Industrial IoT per l'Industria 4.0 (MOMIS I4.0), una applicazione Web e Mobile basata sulle tecnologie più avanzate in ambito Industrial IoT, Big Data, Intelligenza Artificiale (AI) e Machine Learning che consente di raccogliere e gestire in maniera efficace i Big Data generati da macchinari, reti di sensori dei processi industriali. La piattaforma fornisce strumenti per il monitoraggio continuo e servizi avanzati per l'analisi in tempo reale delle performance di produzione e di qualità. Grazie all'analisi dei dati raccolti, è possibile apprendere dall'esperienza per attuare politiche di manutenzione predittiva, ottimizzare i processi produttivi e ridurre i consumi energetici.
La piattaforma MOMIS Industrial IoT per l'Industria 4.0 è altamente flessibile e configurabile per l'utilizzo in diversi settori produttivi come i settori Energetico, Ceramico, Meccanico, Logistica, Farmaceutico e Biomedicale. Gli strumenti forniti consentono di ottenere i seguenti vantaggi competitivi:

- Sviluppare prodotti e servizi innovativi Industry 4.0 da fornire al mercato
- Migliorare la qualità dei prodotti e dei processi produttivi grazie all'integrazione e l'analisi dei Big Data prodotti dai macchinari
- Ottimizzare i servizi di manutenzione presso i clienti grazie ad allarmi tempestivi sullo stato di funzionamento dei macchinari
- Ridurre i costi di produzione grazie alla diminuzione dei guasti e dei malfunzionamenti inattesi
- Ottimizzare i consumi energetici e ridurre i costi di produzione grazie all'attuazione di politiche di "energy saving"

La piattaforma è stata progettata e realizzata da DataRiver con un approccio modulare come un sistema flessibile e aperto, per permettere future estensioni di nuove funzionalità, nuove modalità di connessione e raccolta dati da impianti produttivi, nuovi dispositivi o nuovi servizi. Grazie alla modularità si presta facilmente anche all'integrazione con altri sistemi software presenti nelle aziende o a soluzioni di terze parti. MOMIS Industrial IoT è un'applicazione web multipiattaforma progettata per l'utilizzo da parte degli utenti finali sia da PC che da dispositivi mobili (Smartphone e Tablet).

2. Stack Tecnologico

La Figura 1 mostra lo stack tecnologico della piattaforma MOMIS Industrial IoT per l'Industria 4.0, usata come base per la Energy Community Data Platform, pensato per sfruttare le tecnologie opensource in sinergia con il sistema Open Source MOMIS (Mediator EnvirOnment for Multiple Information Sources) che adotta un approccio semantico per l'integrazione delle sorgenti dati e facilita l'acquisizione di nuove sorgenti.

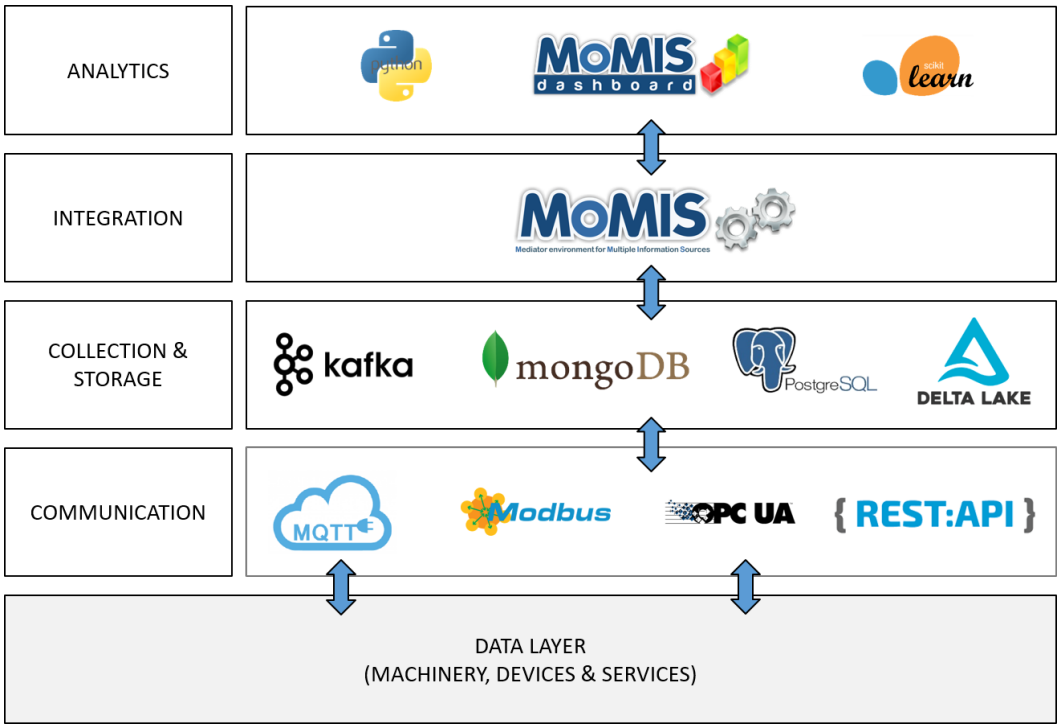


Figura 1 – Stack tecnologico Piattaforma MOMIS Industrial IoT

La piattaforma consente la raccolta dati (livello Data Layer) che possono provenire da macchinari, dispositivi e servizi web oppure da applicativi aziendali (come CRM o ERP) in forma di sorgenti relazionali (Database) o semi strutturate (ad esempio CSV).
Il livello di comunicazione è realizzato con tecnologie che consentano l'accesso e la raccolta delle informazioni provenienti dal livello sottostante, questi connettori gestiscono sia i principali protocolli industriali (come MQTT e OPC-UA) sia i servizi di comunicazione con le sorgenti dati accessibili tramite REST API.
Il livello di data collection e storage, implementa tecnologie che consentono l'ottimizzazione del flusso dati (Apache Kafka) e lo storage con soluzioni ibride pensate per integrare sia soluzioni NoSQL (MongoDB, Delta Lake), fortemente orientate all'immagazzinamento e storicizzazione di grandi volumi di dati, che relazionali (come PostgreSQL), invece fortemente orientate al processamento e all'interrogazione dei dati.

Il livello di integrazione implementa la tecnologia del sistema MOMIS che è in grado di aggregare e rendere omogenee informazioni provenienti da sorgenti dati eterogenee, sia strutturate che semi-strutturate, in modo semi-automatico.
Il livello di analisi dei dati utilizza tecnologie per la data preparation e Machine Learning (Scikit-learn, Apache Spark) per poter interagire con la soluzione per la data analytics basata sul MOMIS, ovvero MOMIS Dashboard.

3. Architettura

L'architettura mostrata in Figura 2, rappresenta i moduli funzionali della versione della Energy Community Data Platform proposta in fase di analisi del progetto:

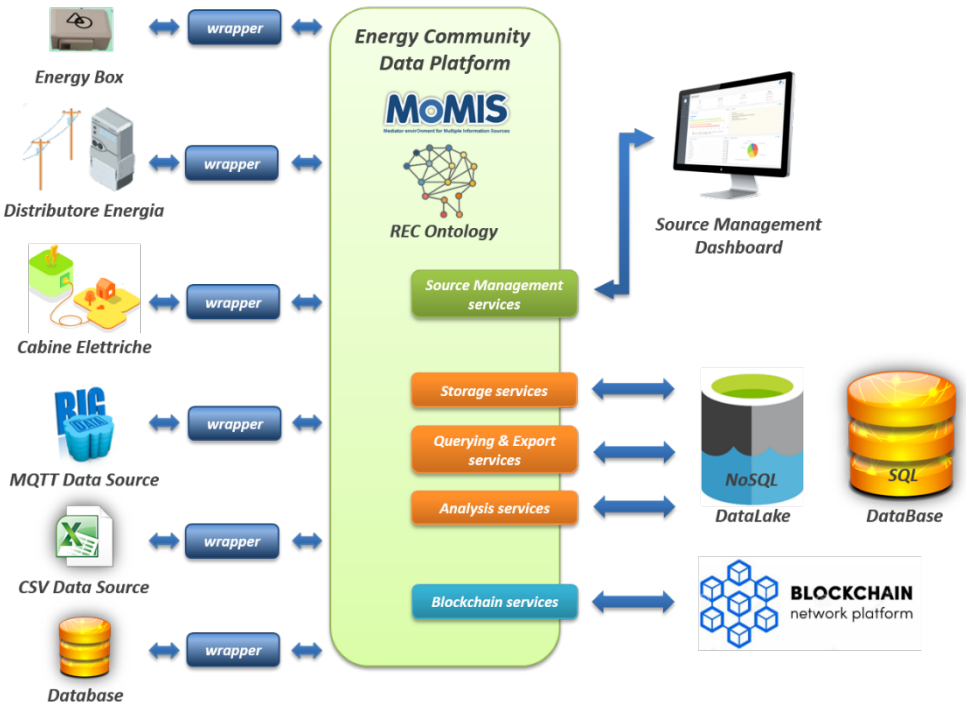


Figura 2 – Proposta Architettura Piattaforma MOMIS Industrial IoT

L'architettura nel corso del progetto si è evoluta per tenere conto delle esigenze espresse da ENEA e dalla reale struttura delle Sorgenti Dati disponibili per il progetto. Infatti al contrario di quanto ipotizzato in fase di definizione dei requisiti del progetto, i dati non vengono inviati direttamente da Energy Box, distributori energia e cabine elettriche ma repository FTP regolarmente alimentati che si vanno ad aggiungere alle sorgenti relazionali già definite anche nella versione iniziale. Inoltre per poter gestire in maniera efficiente i flussi è stato introdotto anche una soluzione per il coordinamento delle operazioni sui dati basata su Airflow, come mostrato in figura:

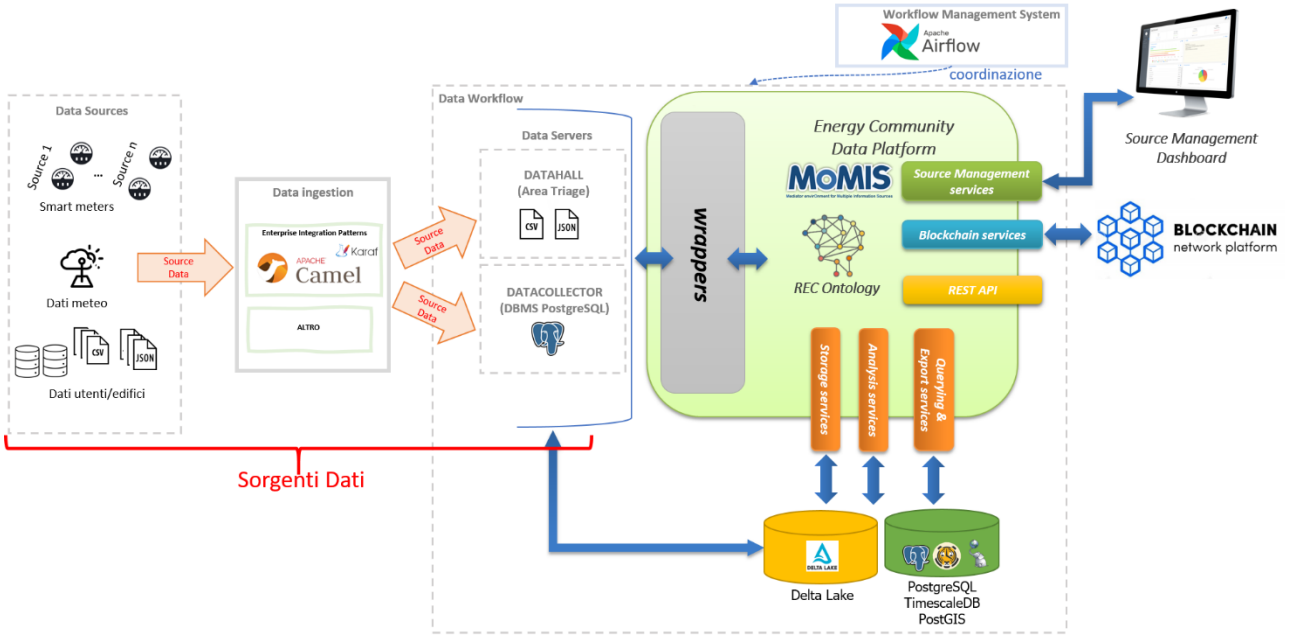


Figura 3 – Architettura finale Piattaforma MOMIS Industrial IoT

Di seguito vengono descritti i moduli software e servizi che vanno a costituire l'architettura illustrata in precedenza.

3.1. Wrapper

Il servizio di ingestion, che è costituito da moduli software chiamati wrapper, consente di connettere i dispositivi, servizi e in generale le sorgenti di dati. Tramite questo servizio vengono estratti gli schemi e le caratteristiche delle sorgenti dati da integrare nel sistema. Le interfacce create dall'analisi degli schemi, consentono la rappresentazione delle sorgenti eterogenee (sia per struttura che per protocollo utilizzato) in un linguaggio comune che, tramite i servizi di integrazione, viene uniformate. Per la piattaforma Energy community Data Platform, sono stati creati wrapper per gestire i flussi dati relativi a:

- **Dati condominiali**: dati con granularità al secondo che devono essere sincronizzati con cadenza settimanali. Per questo flusso dati è stata sviluppata una versione del wrapper CSV che è in grado di connettersi al server FTP di ENEA chiamato "Data Hall", individuazione degli archivi ZIP contenenti i dati, estrarre e ricavare la rappresentazione dei file CSV contenuti negli archivi

- **Dati meteo**: dati con granularità al minuto contenenti le informazioni raccolte dalle centraline meteo. Per questo flusso è stata sviluppata una versione del wrapper JSON che è in grado di connettersi al server "Data Hall", cercare i file JSON e ricavare cavarne la rappresentazione

- **Dati ARERA**: dati con granularità al quarto d'ora contenti le informazioni relative alle singole utenze elettriche dotate di contatore di nuova generazione. Per questo flusso è stata sviluppata una versione del wrapper CSV che è in grado di connettersi al server "Data Hall", cercare i file CSV e ricavarne la rappresentazione

- **Dati Letture del Database SelfUser**: dati con granularità al quarto d'ora relativi alle singole utenze elettriche, dati meteorologici e dati sui consumi delle palazzine. Per questo flusso è stato utilizzato il wrapper PostgreSQL

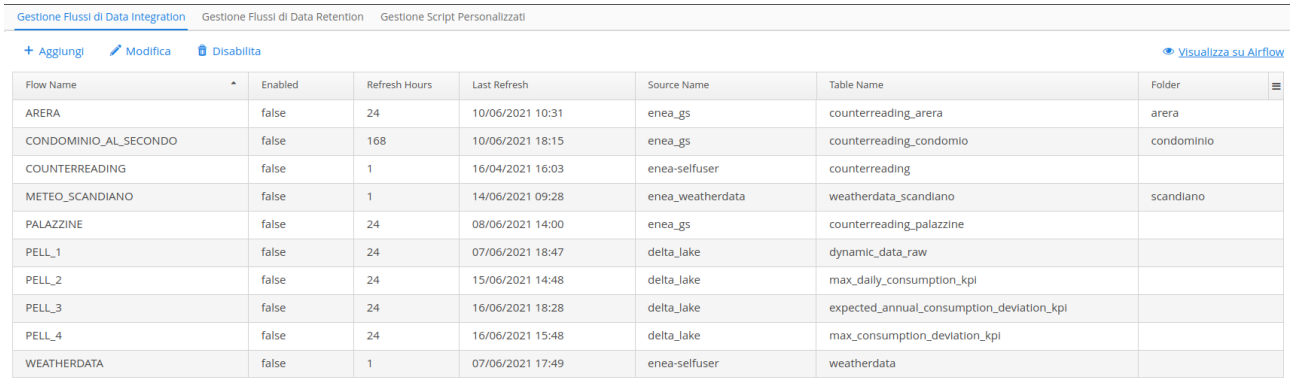
- **Dati PELL:** dati già elaborati da ENEA contenenti dei KPI di interesse relativi al caso d'uso del PELL ovvero relativi all'illuminazione pubblica, fanno sempre parte delle letture ma hanno una gestione

differente. I dati vengono salvati sul data lake realizzato con tecnologia Delta Lake. Per questo flusso dati è stato sviluppato il wrapper Delta Lake, che è in grado di connettersi alla sorgente big data e ricavarne la rappresentazione del dato.

3.2. Integration (MOMIS Mediator)

Il modulo software di data integration realizzato sulla base del sistema a mediatore MoMIS provvede alla rappresentazione semantica delle sorgenti dati consentendo la creazione di una ontologia relativa al dominio applicativo della piattaforma, Renewable Energy Community (REC) Ontology, che rappresenta in modo omogeneo tutte le informazioni raccolte. Grazie all'ontologia definita dai servizi di integrazione è possibile estendere il sistema a nuove sorgenti dati, anche di tipologie differenti. L'integrazione semantica infatti consente alla piattaforma di ricavare in maniera trasparente il mapping fra sorgenti e schema di destinazione.
In questo progetto sono stati sviluppati i servizi di supporto all'integrazione che consentono di configurare le funzioni di calcolo da applicare ai flussi di dati dei POD, di poter aggiungere facilmente nuovi flussi di dati, che vengono così integrati e resi omogenei allo schema di destinazione. Le operazioni di trasformazione sviluppate per questo progetto sono: applicazione di operatori matematici, calcolo di medie, massimi, minimi, conversione temporale in vari formati, tra i quali UNIX, inserimento di valori mancanti su base temporale, aggiunta ed scarto di tuple sulla base del timestamp dei dati.

La Figura 4 mostra l'interfaccia grafica per la chiamata dei servizi di integrazione:



Flow Name	Enabled	Refresh Hours	Last Refresh	Source Name	Table Name	Folder
ARERA	false	24	10/06/2021 10:31	enea_gs	counterreading_arera	arera
CONDOMINIO_AL_SECONDO	false	168	10/06/2021 18:15	enea_gs	counterreading_condomio	condominio
COUNTERREADING	false	1	16/04/2021 16:03	enea-selfuser	counterreading	
METEO_SCANDIANO	false	1	14/06/2021 09:28	enea_weatherdata	weatherdata_scandiano	scandiano
PALAZZINE	false	24	08/06/2021 14:00	enea_gs	counterreading_palazzine	
PELL_1	false	24	07/06/2021 18:47	delta_lake	dynamic_data_raw	
PELL_2	false	24	15/06/2021 14:48	delta_lake	max_daily_consumption_kpi	
PELL_3	false	24	16/06/2021 18:28	delta_lake	expected_annual_consumption_deviation_kpi	
PELL_4	false	24	16/06/2021 15:48	delta_lake	max_consumption_deviation_kpi	
WEATHERDATA	false	1	07/06/2021 17:49	enea-selfuser	weatherdata	

Figura 4 – UI servizi di integrazione

3.3. Storage

I servizi di storage si occupano di memorizzare i dati a seconda della tipologia e delle caratteristiche contenute nella descrizione della sorgente dei dati. Questi servizi comunicano con il modulo di integrazione per conoscere la descrizione e le caratteristiche dei dati che saranno memorizzati. Lo storage per la piattaforma è di tipo ibrido ovvero realizzato con database relazionali basati su una versione di PostgreSQL ottimizzata per le analisi temporali, tramite il plugin TimescaleDB e PostGIS, e non relazionali realizzato con la tecnologia Delta Lake che consente di realizzare dei Data Lake con le caratteristiche di transazionalità tipiche dei database relazionali.
In questo progetto le 2 soluzioni di storage sviluppate hanno destinazioni di utilizzo distinte:

Delta Lake
Il Data Lake con tecnologia Delta è il punto di storage di big data, e pensato per delle analisi offline sui dati.

Il Delta Lake è anche utilizzato per la storicizzazione di dati che non hanno necessità di essere consultati in real-time, infatti tramite il servizio di storage, viene verificata la marca temporale del dato e quando il dato supera il periodo di retention viene storicizzato nel Delta Lake e contestualmente eliminato dal database PostgreSQL.

PostgreSQL con PostGIS e Timescale
Il database relazionale PostgreSQL ottimizzato per le interrogazioni su serie temporali che consente di eseguire le interrogazioni sui dati più aggiornati. Il database relazionale è in grado di avere prestazioni superiori alle altre fonti di dati, per questo motivo viene utilizzato per rispondere alle richieste più frequenti (tipicamente sui dati più aggiornati).
Oltre al dato aggiornato contiene anche dei dati storici di sintesi provenienti dal Delta Lake.

3.4. Querying e Export

I servizi di interrogazione ed esportazione dei dati si occupano di consentire l'accesso alle diverse tipologie di dati presenti all'interno del sistema, agli utenti ed applicazioni autorizzate per l'accesso ad una determinata porzione di dati o ad una determinata vista di aggregazione dei dati. L'accesso ai dati e ai diversi servizi implementati nella piattaforma è gestito attraverso un sistema di controllo degli accessi basato su ruoli e permessi (Role Based Access Control) che possa garantire ad ogni tipologia di utente il diritto di accesso alle informazioni di cui ha bisogno e al tempo stesso garantire la sicurezza delle informazioni a cui non ha diritto ad accedere, in base al ruolo che l'utente ha all'interno di una organizzazione.

Per la piattaforma Energy Community Data Platform sono stati realizzati 2 API distinte

- **ExportUD:** consente di interrogare la tabella integrata delle letture e di esportare i valori in formato UrbanDataSet-XML, UrdanDataSet-JSON e in formato grezzo CSV.
- **SQL_API**: consente di eseguire le query utente, sulla base delle query definite nell'applicazione e dei relativi permessi di accesso. Per dare la massima flessibilità richiesta da ENEA, per i parametri non vi sarà alcun tipo di controllo di integrità. Questa implementazione può essere soggetta ad attacchi di SQL Injection e quindi causa di un fallimento durante un penetration test. Il passaggio di parametri liberi costituisce un possibile punto di attacco malevolo.

3.5. Scheduling di Script Customizzati

Il servizio di scheduling di script customizzati consente di eseguire uno script sul server di Momis. Il servizio è estremamente flessibile in quanto raccoglie script di diversa natura tramite interfaccia grafica e li schedula secondo la configurazione definita sempre tramite interfaccia.
Questo servizio non controlla cosa contengono gli script e le librerie caricate a supporto dello script ma si limita ad eseguire la classe main (in linguaggio bash o matlab) secondo la configurazione definita.

3.6. Source Management Dashboard

I servizi per la gestione delle sorgenti dati e dello storage consentono il monitoraggio dei dati raccolti e processati dagli altri moduli software, fornendo indicatori chiave dei dati e dei processi che sono visualizzabili su mappe, tabelle aggregate e grafici, personalizzati in base alle esigenze specifiche dell'utente e del progetto. Il servizio è pensato per essere facilmente estensibile ed estendibile per integrare informazioni provenienti da sorgenti dati esterne. Per la realizzazione della Source Management Dashboard

è stato utilizzato il modulo di Data Analytics di MOMIS, chiamato anche MOMIS Dashboard. Le principali funzionalità della Source Management Dashboard sono le seguenti:

- Visione unificata dei dati aziendali integrati con sorgenti dati esterne
- Ricerca e monitoraggio di dati aggregati provenienti da fonti dati distribuite ed eterogenee
- Visualizzazione di indicatori su grafici e tabelle dinamiche
- Gestione sicurezza e visibilità dei dati basata su ruoli e gruppi di utenti

4. Sviluppo moduli di Energy Community Data Platform

4.1. Wrapper

Per il modulo wrapper sono state definite insieme ad ENEA le politiche e le caratteristiche dei flussi di dati che i wrapper consentono di acquisire interpretare e inviare ai servizi di integrazione. Di seguito vengono descritte le proprietà dei wrapper sviluppati nella piattaforma indicando a quale sorgente accedono e come devono gestire i dati.

4.1.1. Dati dell'Area Triage

Per i dati presenti nel repository di ENEA definito come Area Triage, è stato realizzato un wrapper per l'accesso tramite protocollo FTP ai file CSV o JSON provenienti dalla data ingestion di Apache Camel e Apache Karaf.

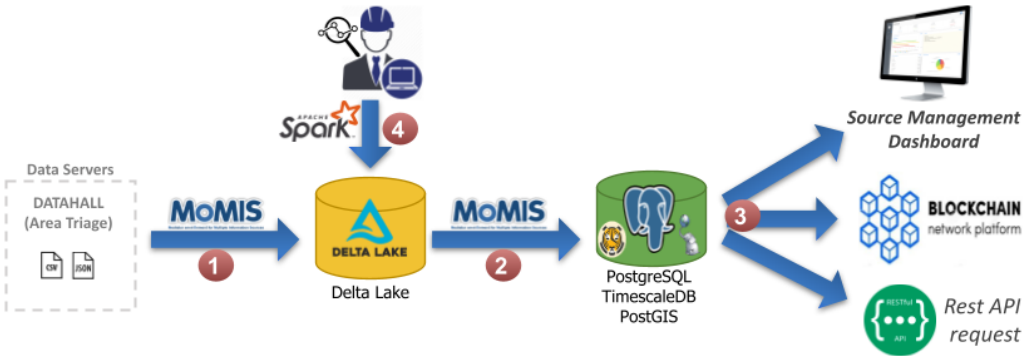


Figura 5 - Schema flusso dei dati Area Triage

Tramite i wrapper i dati provenienti dall'Area Triage vengono letti da MOMIS e caricati nel Delta Lake in forma di dato grezzo (nella Figura 5 punto **1**). Parallelamente al salvataggio del dato grezzo verranno eseguite elaborazioni del dato stesso per:

- Aggiornamento dei livelli più raffinati del Delta Lake con sintesi dei dati progressive.
- Sintesi del dato da materializzare sul PostgreSQL (nella Figura 5 punto **2**) per essere disponibile per Source Management Dashboard, API (nella Figura 5 punto **3**). NOTA nel progetto era prevista anche l'esportazione dei dati come blockchain, ma questa funzionalità è stata demandata a progetti futuri

È stata aggiunta in questo progetto anche la possibilità di accesso tramite Spark al Delta Lake per consentire ai Data Analyst altre tipologie di analisi sui dati grezzi (nella Figura 5 punto **4**).

Questa gestione del dato è stata implementata per i flussi:

- Flusso EnelX con dati al secondo

Questo flusso contiene i dati di consumo/produzione raccolti da appartamenti privati e condominio.
Location: datahall/enelx/Condominio*.zip
Formato: CSV
Granularità: dati al secondo
Sincronizzazione: settimanale

Mappatura dei campi:

Campo del CSV raggruppati per 15 minuti	Campo del PostgreSQL
Sensor_ID	counterreading.deviceID
-	counterreading.flowID = 5
Local_Time	counterreading.timestamp
AVG(CurrentA)	counterreading.current
AVG(VoltageV)	counterreading.voltage
AVG(Pow_activeW)	counterreading.powactive
AVG(Pow_reactiveVar)	counterreading.powreactive
AVG(Pow_activeW)**/4**	counterreading.energy

Struttura sul Delta Lake
Livello bronze: dato grezzo con riferimento al flowID aggiunto nel processo di raccolta dati.
Livello silver: aggregazione dei dati che viene gestita con lo storage service per la scrittura nel PostgreSQL con retention a 2 anni.
Livello gold: Nessuno

- Flusso dei dati meteo delle centraline di Scandiano (RE)

Questo flusso di dati contiene le informazioni provenienti dalle centraline meteo.
Location: datahall/centraline_meteo/*.json
Formato: JSON
Granularità: dati al minuto

Mappatura dei campi:

Campo del JSON	Campo del PostgreSQL
ownerName	weatherdata.deviceID
SE currConditionValues.sensorDataName = Temp currConditionValues.convertedValue	weatherdata.temp

Campo del JSON	Campo del PostgreSQL
SE currConditionValues.sensorDataName = Hum currConditionValues.convertedValue	weatherdata.humidity
Nome del file	weatherdata.timestamp
-	FlowID = 7

Struttura sul Delta Lake
Livello bronze: il dato grezzo con riferimento al flowID aggiunto nel processo di raccolta dati.
Livello silver: aggregazione dei dati che viene gestita con lo storage service per la scrittura nel PostgreSQL con retention a 2 anni.
Livello gold: Nessuno

- Flusso Dati ARERA

Questo flusso contiene i dati di consumo al quarto d'ora relativi alle singole utenze elettriche dotate di contatore di nuova generazione.
Location: datahall//enea_community/Portale_Consumi_ARERA_raw/*.csv
Formato: CSV
Granularità: dati al quarto d'ora

Mappatura dei campi:

Campo del CSV	Campo del PostgreSQL
Pod	counterreading.deviceID
data_lettura	counterreading.timestamp
data_ricezione	counterreading.tsi
ea1-ea96	counterreading.energy
er1-er96 (*4)	counterreading.powreactive
-	FlowID = 3

Struttura sul Delta Lake
Livello bronze: dato grezzo con riferimento al flowID aggiunto nel processo di raccolta dati.
Livello silver: aggregazione dei dati che viene gestita con lo storage service per la scrittura nel PostgreSQL con retention a 2 anni.
Livello gold: Nessuno

4.1.2. Dati del Data Collector

Il Data Collector è la sorgente relazionale con DBMS PostgreSQL utilizzata da ENEA per salvare le informazioni gestionali, dati dei consumi provenienti dalle palazzine del centro ENEA di Bologna e del caso Self User, DB che fornisce dati reali appositamente utilizzato per questa attività per poter fare sviluppo e test. A livello implementativo nel Data Collector sono presenti le viste che puntano anche ad altri database di ENEA e che vengono accedute dai wrapper di MOMIS. Di seguito sono descritti i dettagli per ciascun flusso.

- Dati gestionali - DB Selfuser

La sorgente Selfuser contiene dati di profilazione degli utenti, dettagli sui flussi, device, unità di misura e altre informazioni che arricchiscono i dati relativi alle letture ma che hanno un aggiornamento più sporadico rispetto alle letture stesse.

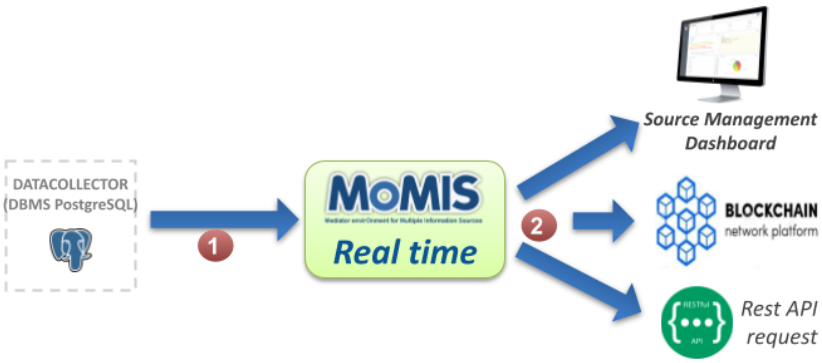


Figura 6 – Schema flusso dei dati gestionali di Selfuser

1. In questo caso per evitare la replicazione dei dati, MOMIS effettua la lettura real-time dal Data Collector senza andare a materializzare le informazioni nel Delta Lake o nel DB PostgreSQL di MOMIS.
2. I dati sono letti al momento della richiesta da parte delle Source Management Dashboard o dalle API.

Non verranno salvate copie intermedie vista la quantità non elevata di dati.

Tabelle accessibili da MOMIS: users, users_communities, users_roles, communities, devices, flows, flows_communities, measurementstables, suppliers, suppliers_communities, t*

- Comunità virtuale uffici e abitazioni: DB Selfuser

Questo flusso contiene i dati dei consumi provenienti dalle palazzine del centro ENEA di Bologna relativi a uffici e abitazioni.

Tabelle accessibili da MOMIS: per le utenze sono le stesse tabelle nel caso di Data Collector: users, users_communities, users_roles, communities, devices, flows, flows_communities, measurementstables, suppliers, suppliers_communities, t*

Le utenze degli uffici sono: edificioA e edificioB …
Le utenze abitazioni sono: utenteA –utenteB ….

Per questi flussi l'integrazione consiste nell'accesso in sola lettura ai dati senza alcuna elaborazione dei dati interrogati.

- Dati letture - DB Selfuser

Questo flusso contiene le informazioni relative ai dati di consumo al quarto d'ora relativi alle singole utenze elettriche, dati meteorologici e dati sui consumi delle palazzine A e B.
Per questi dati MOMIS provvede all'accesso, acquisizione e materializzazione dei dati come indicato nella Figura 7:

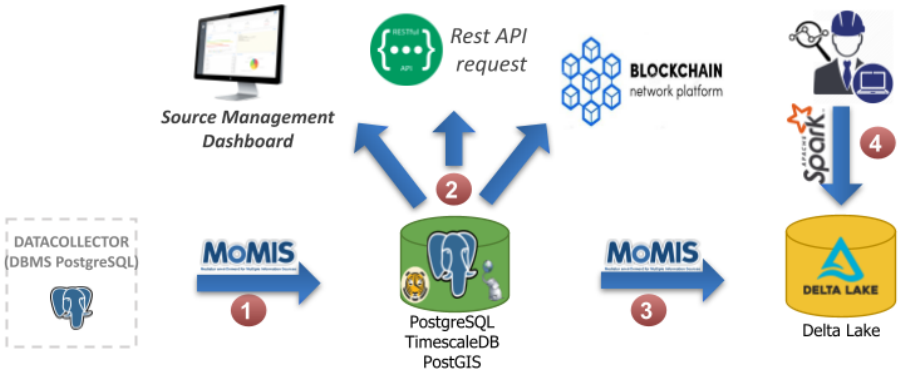


Figura 7 – Schema flusso dei dati letture del Selfuser

1. MOMIS acquisisce il dato dal Data Collector e salva una versione ottimizzata sul suo database PostgreSQL. Il dato può rimosso dal Data Collector per mantenerlo performante.
2. Le informazioni materializzate sul PostgreSQL di MOMIS sono visualizzate sulla Source Management Dashboard e API.
3. L'ultimo step di questo flusso è la materializzazione sul Delta Lake: quando il dato diviene storico, viene rimosso dal PostgreSQL per essere storicizzato sul Delta Lake.
4. Il dato storico è sempre disponibile per l'analisi tramite connettori Spark sul Delta Lake.

4.1.2.1. Weatherdata e Counterreading

Tabelle: weatherdata, counterreading
Sincronizzazione: 15 minuti

Struttura sul Delta Lake
Livello bronze: Saranno i dati che hanno passato i 18 mesi (periodo di data retention)
Livello silver: No
Livello gold: No

4.1.2.2. Consumo elettrico palazzine A e B

Tabelle: Le foreign tables del database consumi

Le tabelle dei consumi sono **delle foreign_tables**:

- powcdza_for: consumi per condizionamento palazzo A (utente: edificioA)
- powfma_for: consumi per forza motrice palazzo A
- powilla_for: consumi per illuminazione palazzo A
- powcdzb_for: consumi per condizionamento palazzo B (utente: edificioB)
- powfmb_for: consumi per forza motrice palazzo B
- powillb_for: consumi per illuminazione palazzo B

Mappatura dei campi:

Campo del DB Consumi	Campo del PostgreSQL
timestamp	counterreading.timestamp
activepow	counterreading.powactive
	flowID = 9 (signalmixarcoveggio)
	device_id = cnda cdnb fma fmb illa illb *dipendono dalla vista che raccoglie i dati*

Sono dati al quarto d'ora quindi in fase di integrazione non è prevista la sintesi del dato.
Sincronizzazione: giornaliera

Struttura sul Delta Lake
Livello bronze: sarà il dato che ha passato i 18 mesi
Livello silver: No
Livello gold: No

- Dati relativi al caso d'uso del PELL

I dati relativi all'illuminazione pubblica, fanno sempre parte delle letture ma hanno una gestione differente. Per questi dati MOMIS provvede all'accesso, acquisizione e materializzazione dei dati come indicato nella Figura 8:

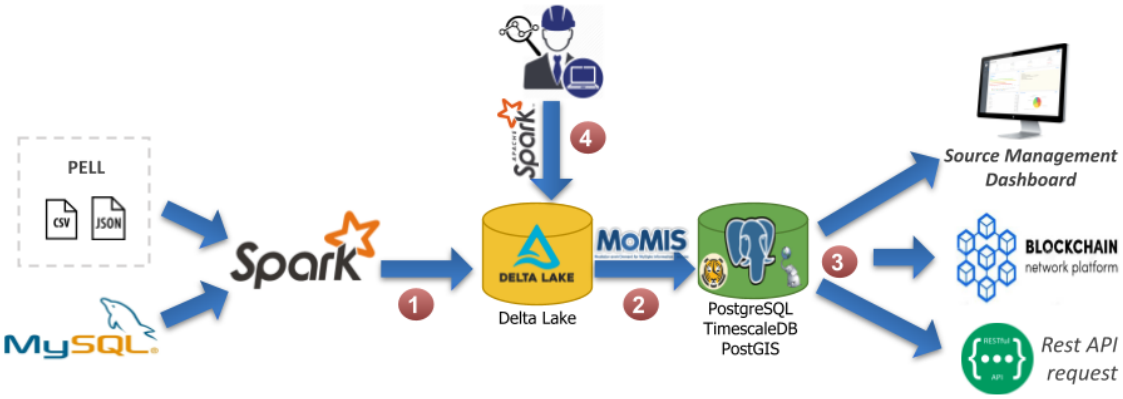


Figura 8 – Schema flusso dei dati PELL

1. I dati in ingresso vengono elaborati da Spark, salvati sul Delta Lake in diversi formati (dati grezzi e altri livelli di elaborazione) e vengono messi a disposizione di MOMIS.
2. MOMIS legge il dato dal Delta Lake e ne salva una versione ottimizzata sul PostgreSQL.
3. Le informazioni materializzate sul PostgreSQL di MOMIS sono visualizzate sulla Source Management Dashboard e API.
4. Il dato storico è sempre disponibile per l'analisi tramite connettori Spark sul Delta Lake

Struttura DB SQL PELL
Schema: PELL
Tabelle:

- Counterreading: Tabella contenente tutte le misure derivanti dall'UD CounterReading, materializzato nel delta lake come "dynamic_data_raw"
- Kpivalues: Tabella contenente i valori dei kpi
- Kpidescriptions: Tabella contenente le descrizioni dei kpi
- t_frequencies: Lookup table per le frequenze di calcolo dei kpi

Poiché si tratta di tabelle nuove e con una struttura differente da quelle presenti sul Data Collector viene mostrata la struttura delle tabelle del DB SQL del PELL.

Tabella counterreading:

Nome	Tipo	Primary	NotNull	Note	Esempio
counterreadingid	bigint	x	x	autoincrement	1,2,3…
producerID	VarChar(20)				Utilityname1,utilityname2 …
timestamp	timestamp			E' possibile accorpare la timezone nel timestamp in fase di inserimento su postgres?	2021-05-12T12:00:00
timezone	VarChar(8)				UTC+1
lat	Double				42.12
lon					12.56
start_period	timestamp			Come timestamp	2021-05-12T13:00:00
end_period	timestamp			Come timestamp	2021-05-12T13:15:00
podID	VarChar(12)				IT001E13171384
electricalpanelID	VarChar(20)				006-0

Nome	Tipo	Primary	NotNull	Note	Esempio
towncode	VarChar(8)				12345678
activeenergy	double				22.34
totalactivepower	double				66.89
currentline1	double				11.2
currentline2	double				12.1
currentline3	double				12.3

Tabella kpivalues:

Nome	Tipo	Primary	NotNull	Note	Esempio
kpivaluesID	bigint	x	x	autoincrement	1
datestart	date			Data di inizio di validità del KPI. KPI annuale: il giorno 01-01 dell'anno corrente, KPI giornaliero: il giorno corrente	2021-01-01,2021-05-12
dateend				Data fine validità del KPI. KPI annuale: 31-12 anno corrente KPI giornaliero: giorno corrente	
podID	VarChar(12)				IT001E13171384
electricalpanelID	VarChar(20)				006-0
value	double				1.12

Tabella kpidescriptions:

Nome	Tipo	Primary	NotNull	Note	Esempio
kpidescriptionID	bigint	x	x	autoincrement	
name	text				"Maximum daily

Nome	Tipo	Primary	NotNull	Note	Esempio
					consumption”
descrizione	Text			Descrizione del KPI	
frequencyID	int			In chiave esterna con l’id della tabella t_frequencies	

Tabella t_frequencies:

Nome	Tipo	Primary	NotNull	Note	Esempio
frequencyID	int	x	x	autoincrement	1,2,3
name	VarChar(10)				“daily”,”yearly”,”monthly”…

DeltaLake della sorgente PELL

Il Mapping tra Delta Lake di ENEA e tabelle PostgreSQL è stato realizzato per mantenere i dati grezzi contenuti in pell_ip_dynamic_data_raw in una tabella strutturalmente identica in PostgreSQL. Per garantire la mappatura tra le tabelle Delta Lake (dedicate ai KPI) e la relativa tabella unica in PostgreSQL, è stata aggiunta l’informazione dell’id. Entrambe le sorgenti (Delta Lake e PostgreSQL) hanno la medesima struttura dati.

Tabelle: pell_ip_dynamic_data_raw, max_daily_consumption_kpi, expected_annual_consumption_deviation_kpi, max_consumption_deviation_kpi

Campo di pell_ip_dynamic_data_raw	Campo del PostgreSQL
producerID	counterreading.producerID
timestamp	counterreading.timestamp
timezone	counterreading.timezone
lat	counterreading.lat
lon	counterreading.lon
start_period	counterreading.start_period
end_period	counterreading.end_period
podID	counterreading.podID
ElectricalPanelID	counterreading.electricalpanelID

Campo di pell_ip_dynamic_data_raw	Campo del PostgreSQL
TownCode	counterreading.towncode
ActiveEnergy	counterreading.activeenergy
TotalActivePower	counterreading.totalactivepower
CurrentLine1	counterreading.currentline1
CurrentLine2	counterreading.currentline2
CurrentLine3	counterreading.currentline3

Campo di max_daily_consumption_kpi	Campo del PostgreSQL
-	kpivalues.kpidescriptionID = 1
datestart	kpivalues.datestart
dateend	kpivalues.dateend
podID	kpivalues.podID
ElectricalPanelID	kpivalues.electricalpanelID
value	kpivalues.value

Campo di expected_annual_consumption_deviation_kpi	Campo del PostgreSQL
-	kpivalues.kpidescriptionID = 2
datestart	kpivalues.datestart
dateend	kpivalues.dateend
podID	kpivalues.podID
ElectricalPanelID	kpivalues.electricalpanelID
value	kpivalues.value

Campo di max_consumption_deviation_kpi	Campo del PostgreSQL
-	kpivalues.kpidescriptionID = 3
datestart	kpivalues.datestart

Campo di max_consumption_deviation_kpi	Campo del PostgreSQL
dateend	kpivalues.dateend
podID	kpivalues.podID
ElectricalPanelID	kpivalues.electricalpanelID
value	kpivalues.value

Sincronizzazione (intervallo di aggiornamento dei dati sul PostgreSQL): sincronizzazione giornaliera

4.2. Integration

Per il modulo di software di integrazione dati è stato realizzato lo schema integrato della piattaforma, che descrive le regole di mappattura tra le sorgenti acquisite tramite i wrapper e i database di MOMIS (relazionale realizzato con PostgreSQL e non relazionale realizzato con tecnologia Delta Lake). Le fasi che hanno portato alla definizione dello schema integrato sono le seguenti:

- Caricamento delle sorgenti
- Annotazione semantica
- Generazione delle relazioni semantiche
- Definizione e raffinamento delle regole di mapping

Di seguito vengono descritte nel dettaglio le varie fasi per la realizzazione dello schema integrato.

- Caricamento delle sorgenti

Per ogni sorgente sono stati utilizzati i wrapper descritti in precedenza per accedere ai dati ed estrarre i metadati relativi a formato dati presenza di attributi chiave sulle sorgenti e la tipologia di dati contenuti nella sorgente. La struttura di ogni sorgente viene tradotta in un linguaggio comune per la modellazione delle sorgenti chiamato ODLI3.

Nella Figura 9 viene mostra un'anteprima dei dati acceduti dal wrapper per la verifica della corretta comunicazione con la sorgente:

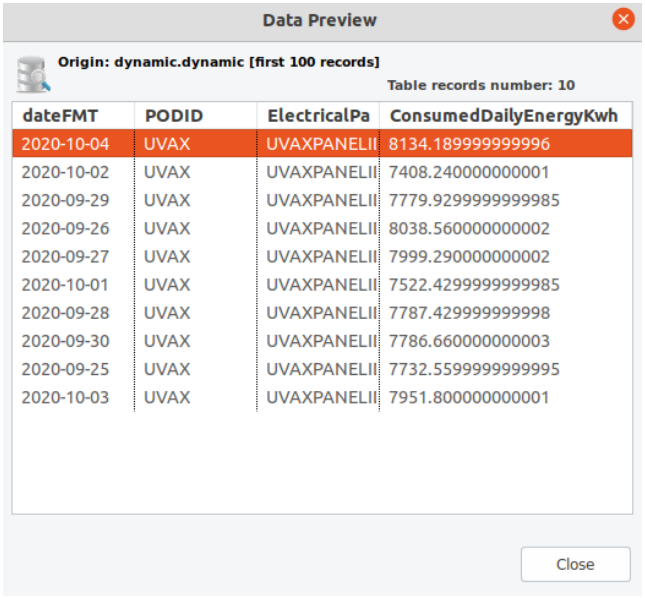


Figura 9 - Integrazione: anteprima dati

- Annotazione dei campi

L'annotazione è una mappatura di un dato termine (nomi di classi e attributi) in un insieme ben definito di concetti di un'ontologia lessicale. Il processo di annotazione consiste nell'associare a ciascun termine uno o più significati relativi ad un comune riferimento lessicale. Grazie alla funzionalità di annotazione semantica di MOMIS sono stati definiti i concetti rappresentati dalle sorgenti ed esteso il riferimento lessicale con i termini specifici del dominio delle energie rinnovabili andando a realizzare una ontologia specifica, ovvero l'ontologia REC.

La Figura 10 mostra il risultato del processo di annotazione dei termini presenti nelle sorgenti acquisite nella fase precedente.

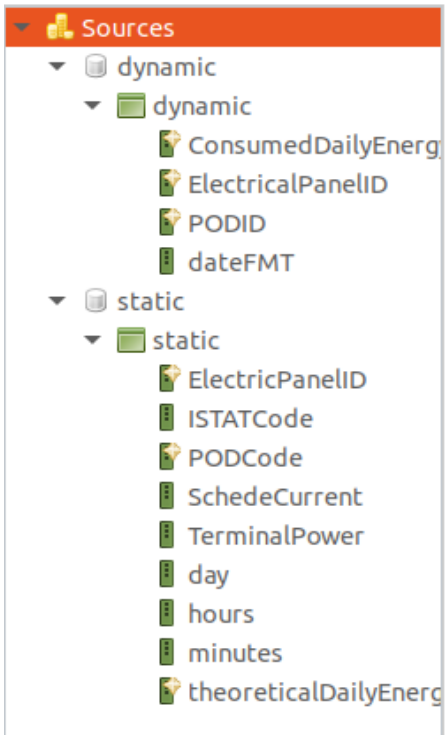


Figura 10 - Integrazione: Annotazione sorgenti dati

Con la annotazione viene assegnata una glossa tra quelle presenti nel thesaurus di MOMIS. Il glossario può essere espanso con i termini specifici per il dominio applicativo dello schema integrato che è stato realizzato.
Ad esempio è stato aggiunto il termine POD (Point of Delivery) con la relativa descrizione (glossa)

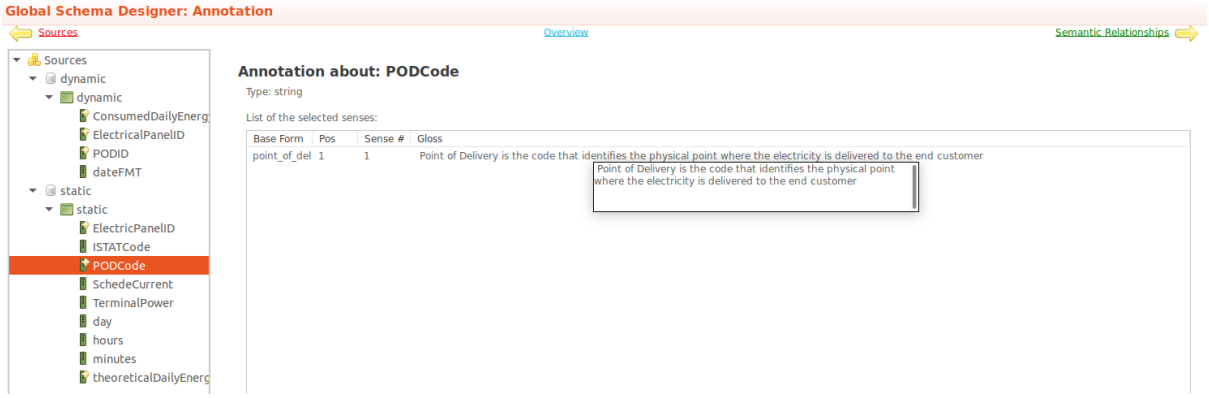


Figura 11 - Integrazione: glossa e lemma di un termine annotato

Per ogni termine annotato è stato anche possibile verificare le relazioni di iperonimia tramite il grafo degli iperonimi. Questo strumento ha aiutato a determinare la corretta annotazione dei termini.

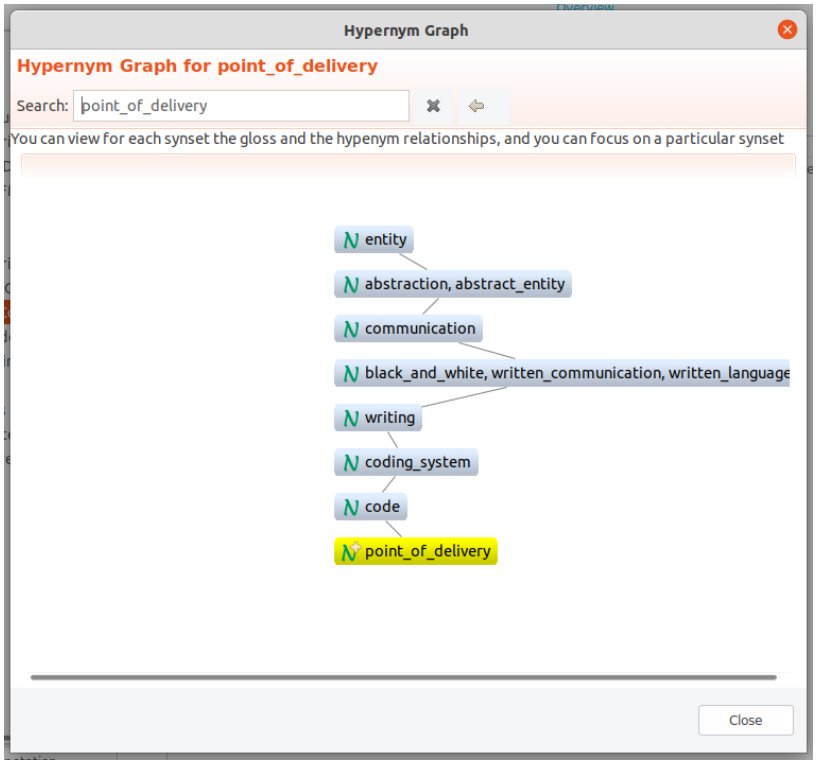


Figura 12 – Integrazione: Grafo degli iperonimi del termine POD

- Generazione delle relazioni semantiche, definizione e raffinamento delle regole di mapping

Le classi e gli attributi appartenenti alle sorgenti che sono coinvolte nell'integrazione e tramite ciò è stato possibile identificare le classi che descrivono gli stessi concetti semantici e attraverso la personalizzazione dei pesi delle relazioni semantiche rilevate, sono state generate le classi e gli attributi globali.

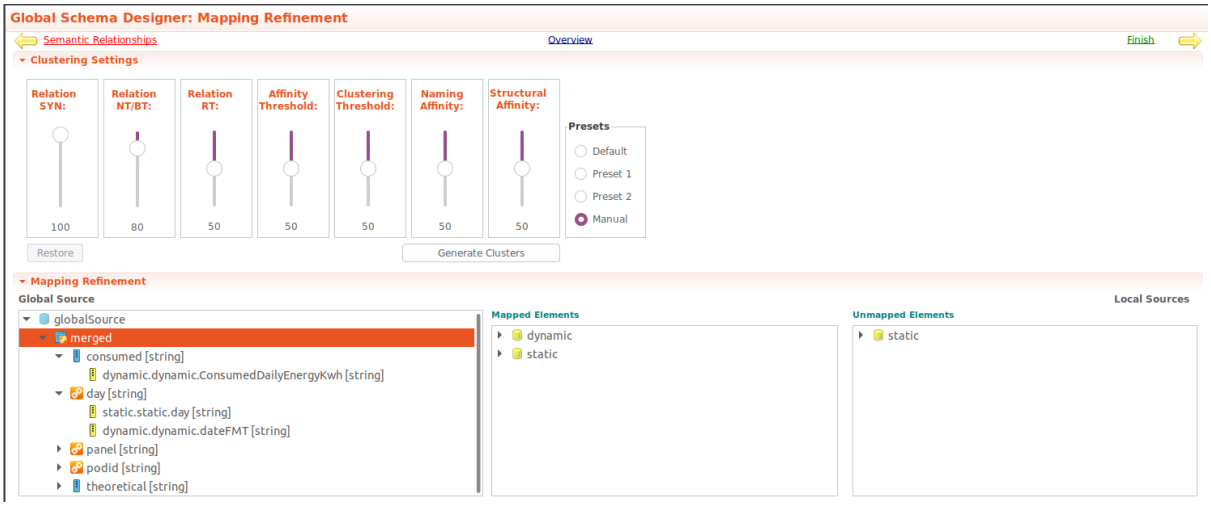
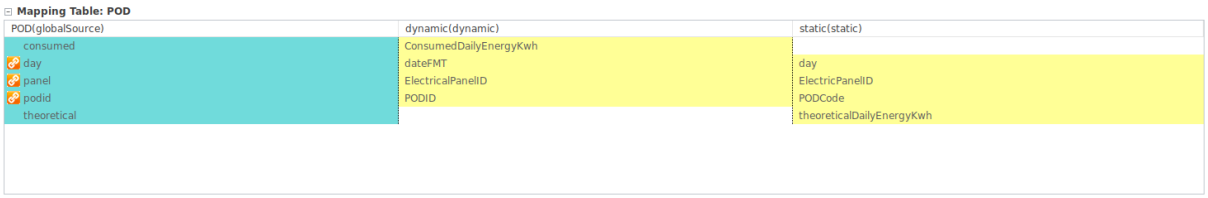


Figura 13 - Integrazione: interfaccia di gestione mapping delle sorgenti

Quali compongono una o più tabelle di mapping tra le sorgenti e lo schema integrato globale:



Le tabelle sono state raffinate andando a definire logiche per la trasformazione dei dati, la fusione e la risoluzione dei conflitti.

Inoltre è stato possibile definire la funzione per il join delle sorgenti ovvero definire gli attributi di join ovvero gli attributi che consentono di identificare le stesse entità nelle diverse sorgenti:

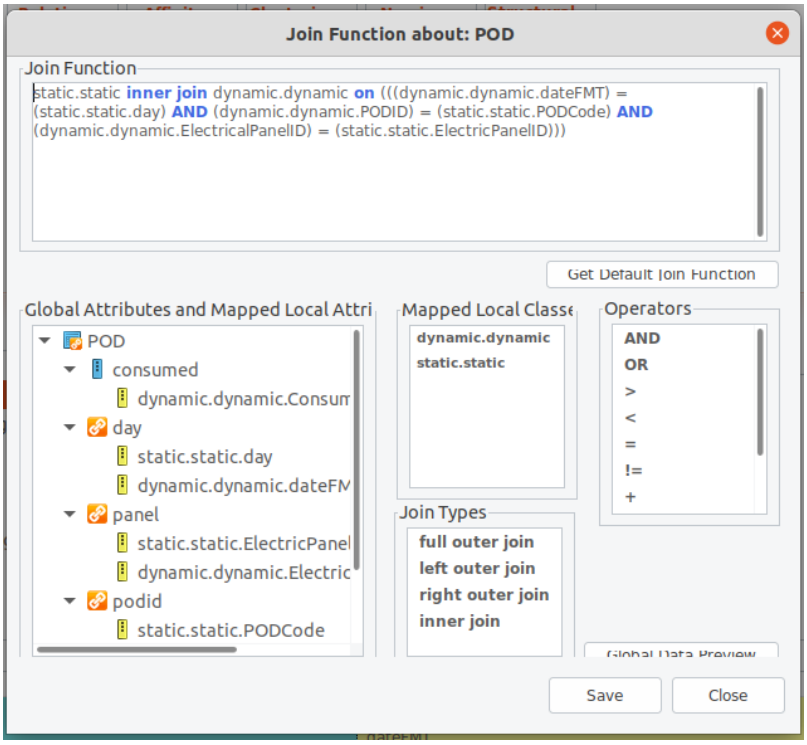


Figura 14 - Integrazione: Definizione regole di JOIN

Questo processo ha generato come output i file di configurazioni di MOMIS che consentono l'integrazione dei dati secondo le regole descritte nei flussi.

- Data Cleaning

Per facilitare la fase di aggiunta di nuove sorgenti dati da integrare, sono state definite e realizzate tutte le funzioni di trasformazione dei dati (operazioni di Data Cleaning) in una versione più completa e più user friendly in modo da poter essere utilizzate anche da utenti non esperti di data integration.

Di seguito viene mostrata la tabella con le funzioni di trasformazione di base presenti in MOMIS:

NOME	DESCRIZIONE
Drop Columns	Operazione che permette l'eliminazione di una lista di colonne.
Filter rows	Operazione che permette di filtrare le righe in base al valore di una colonna. I valori possibili vengono concatenati in una lista.
Drop rows where column has null value	Elimina le righe dove la colonna specificata ha valore nullo.

NOME	DESCRIZIONE
Create or update column by another column	Operazione che permette la creazione o l'aggiornamento di una colonna con i valori di un'altra colonna in base a una condizione. X | Y --- Primo | 22 Secondo | 34 Terzo | 45 ES: Crea una colonna Z mettendo il valore della colonna Y dove X= "Primo" o "Secondo" e il valore della colonna X nelle altre righe. X | Y | Z --- Primo | 22 | 22 Secondo | 34 | 34 Terzo | 45 | Terzo
Create or update column with a value	Operazione che permette la creazione o l'aggiornamento di una colonna con dei valori in base a una condizione. X | Y --- Primo | 22 Secondo | 34 Terzo | 45 ES: Crea una colonna Z mettendo 3 dove X= "Primo" o "Secondo" e 0 altrove. X | Y | Z --- Primo | 22 | 3 Secondo | 34 | 3 Terzo | 45 | 0
Create or update column	Operazione che permette la creazione o l'aggiornamento di una colonna. X | Y --- Primo | 22 Secondo | 34 Terzo | 45 ES: Crea una colonna Z con valore "Nuovo valore". X | Y | Z --- Primo | 22 | Nuovo valore Secondo | 34 | Nuovo valore

NOME	DESCRIZIONE
	Terzo | 45 | Nuovo valore
Sort by column	Ordina le righe in base al valore di una colonna
Drop duplicates by column	Operazione che rimuove i duplicati raggruppando per le colonne selezionate. Se incontra un duplicato la riga che elimina è la prima che incontra. Si consiglia di utilizzare la funzione sort per ordinare le righe prima di eliminare i duplicati. X | Y | Z --- Primo | 22 | Nuovo valore Secondo | 34 | Nuovo valore Primo | 45 | Nuovo valore ES: Elimina le righe doppie raggruppando per la colonna X X | Y | Z --- Secondo | 34 | Nuovo valore Primo | 45 | Nuovo valore
Transpose column	Operazione che permette di trasporre le colonne in righe. Vanno selezionate tutte le colonne i cui valori verranno "duplicati" dalla trasposizione, le colonne da trasporre e il nome delle due colonne nuove. X | Y | Z --- A | 22 | 33 B | 34 | 27 C | 45 | 89 ES: trasporre le colonne Y, Z in un'unica colonna YZ mantenendo fissa la colonna X. X | Old Name | YZ --- A | Y | 22 B | Y | 34 C | Y | 45 A | Z | 33 B | Z | 27 C | Z | 89
Extract file name	Estrae il nome del file e lo salva come valore di una colonna.
Describe	Calcola AVG, il MIN, il MAX e la SUM delle colonne selezionate raggruppando in base alle

NOME	DESCRIZIONE
column	colonne scelte come group by columns. X | Y --- A | 22 A | 44 B | 29 B | 34 C | 45 Descrivi la colonna Y raggruppando per X. X | Y_mean | Y_max | Y_min | Y_sum --- A | 33 | 44 | 22 | 66 B | 31.5 | 34 | 29 | 63 C | 45 | 45 | 45 | 45
Arithmetical Operation	Permette di definire un'operazione aritmetica sulle colonne (+,*, /, ^) X | Y --- A | 22 A | 44 B | 29 B | 34 C | 45 ES: eleva la colonna Y alla seconda X | Y --- A | 484 A | 1936 B | 841 B | 1156 C | 2025
Timestamp To Unix	Converte il timestamp in tempo Unix. Prende in input anche il nome della colonna da generare: se questa non è fra le colonne disponibili ne crea una nuova e permette di non sovrascrivere la precedente. ES: 2020-09-28 01:00:00+02 -> 1601247600
Unix to Timestamp	Procedimento opposto al precedente. Converte il tempo Unix in timestamp. Prende in input anche il nome della colonna da generare: se questa non è fra le colonne disponibili ne crea una nuova e permette di non sovrascrivere la precedente.

NOME	DESCRIZIONE
	ES: 1601247600 -> 2020-09-28 01:00:00+02

E quelle realizzate su richiesta specifica di ENEA:

NOME	DESCRIZIONE	Funzione implementata
AVG	Operazione che aggiunge una colonna con valore calcolato sulla media di un'altra colonna	Describe column
MIN	Operazione che calcola il valore minimo di una colonna	Describe column
MAX	Operazione che calcola il valore massimo di una colonna	Describe column
SUM	Operazione che calcola la somma dei valori di una colonna	Describe column
SUM	Operazione che permetta la somma/sottrazione di una colonna per un valore	Arithmetical Operation
MUL	Operazione che permetta la moltiplicazione di una colonna per un valore	Arithmetical Operation
DIV	Operazione che permetta la divisione di una colonna per un valore	Arithmetical Operation
Time to Unix	Converte una colonna datetime in tempo Unix.	Arithmetical Operation
Add UnixTime	Aggiunge una colonna dove inserisce il tempo Unix, calcolato in base al DateTime di un'altra colonna della stessa riga	Timestamp To Unix
UnixTime to DateTime	Converte il valore UnixTime in Datetime	Unix to Timestamp
Add DateTime	Aggiunge una colonna dove inserisce il tempo in formato Datetime, calcolato in base al tempo Unix preso da un'altra colonna della stessa riga	Unix to Timestamp

NOME	DESCRIZIONE	Funzione implementata
Add Time	Aggiunge (o sottrae) ad una colonna con un valore in datetime, un determinato valore (sempre in tempo). Ad esempio, se ho 03:00:00 e aggiungo un'ora, ottengo 04:00:00	Questa trasformazione è già disponibile utilizzando la conversione a tempo UNIX e aggiungendo con Arithmetical Operation quello che serve
FillWithAverage	Inserisce al posto di un valore che ha una determinata condizione (es è nullo) di una colonna, la media calcolata sui valori della stessa colonna presi dalla prima riga precedente e la prima successiva non nulla. ID - Value 10 - 6 11 – null 8 12 – null 8 13 – null 8 14 - 10	Input= colonna di riferimento e valore che deve soddisfare per essere sostituito (ES: null)
InsertInstant	Inserisce una riga quando il valore (temporale) di una colonna meno il valore nella riga precedente è maggiore di un valore di periodo (indicato). Cioè inserisco una riga quando l'intervallo di tempo fra le due righe è maggiore rispetto ad un riferimento dato. Es. se nella colonna X ho gli istanti di tempo, e X(riga24)- X(riga23) > periodo Allora inserisco una nuova riga con tutti i valori presi dalla riga 23 su delle colonne indicate, e con il valore temporale uguale a quello di X23 + periodo. Gli altri valori restano null.	
DropInstant	Cancello una riga quando il valore (temporale) di una colonna meno il valore nella riga precedente è minore di un valore di periodo (indicato). Cioè tolgo una riga quando l'intervallo di tempo fra le due righe è minore rispetto ad un riferimento dato. Es. se nella colonna X ho gli istanti di tempo, e X(riga24)- X(riga23) < periodo	

NOME	DESCRIZIONE	Funzione implementata
	Allora cancello la riga X24.	

4.3. Storage

Per lo storage service è stata effettuata la revisione del servizio andando così a realizzare nuova versione delle logiche di gestione.
Il servizio è stato riprogettato utilizzare come motore di scheduling Apache Airflow ovvero una piattaforma Open source per creare, pianificare e monitorare i flussi di lavoro in modo programmatico. La nuova interfaccia utente realizzata per la ECD Platform consente di utilizzare Airflow in maniera trasparente all'operatore. L'interfaccia di gestione dei flussi di MOMIS ha consentito di creare flussi di lavoro come grafi aciclici diretti (DAG) delle attività.

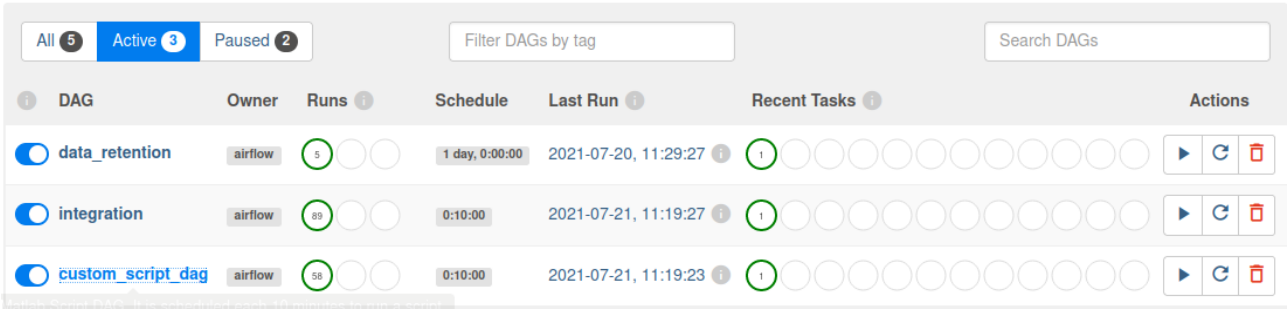


Figura 15 - Storage: dettaglio DAG in Apache Airflow

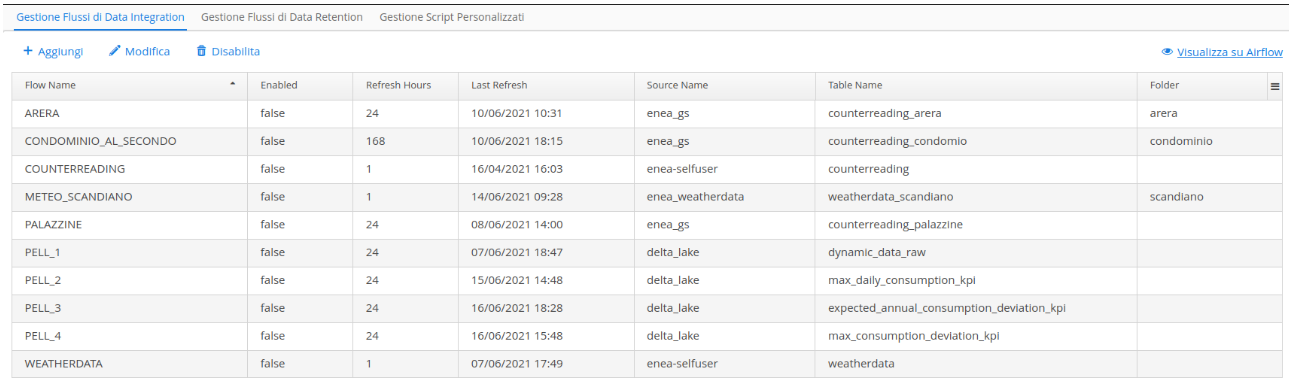
Sono stati elaborati tre differenti DAG con i quali vengono gestite le altrettante funzioni principali di questo progetto:

- **Integration:** ogni 10 minuti controlla se ci sono flussi che necessitano un refresh dei dati e, nel caso, avvia l'integrazione secondo la configurazione
- **Data Retention**: ogni giorno controlla se ci sono flussi che necessitano di un controllo per spostare i dati dalla sorgente relazionale al Delta Lake e, nel caso, avvia il servizio.
- **Custom Script**: ogni 10 minuti controlla se ci sono script che devono essere eseguiti e, nel caso, li esegue.

La configurazione dei singoli flussi e script avviene tramite interfaccia grafica di MOMIS.

- Configurazione dei flussi di integrazione

La Figura 16 mostra l'interfaccia per la visualizzazione e la modifica delle configurazioni dei flussi e la visualizzazione del DAG corrispondente in Airflow.



Gestione Flussi di Data Integration | Gestione Flussi di Data Retention | Gestione Script Personalizzati

+ Aggiungi | Modifica | Disabilita | Visualizza su Airflow

Flow Name	Enabled	Refresh Hours	Last Refresh	Source Name	Table Name	Folder
ARERA	false	24	10/06/2021 10:31	enea_gs	counterreading_arera	arera
CONDOMINIO_AL_SECONDO	false	168	10/06/2021 18:15	enea_gs	counterreading_condomio	condominio
COUNTERREADING	false	1	16/04/2021 16:03	enea-selfuser	counterreading	
METEO_SCANDIANO	false	1	14/06/2021 09:28	enea_weatherdata	weatherdata_scandiano	scandiano
PALAZZINE	false	24	08/06/2021 14:00	enea_gs	counterreading_palazzine	
PELL_1	false	24	07/06/2021 18:47	delta_lake	dynamic_data_raw	
PELL_2	false	24	15/06/2021 14:48	delta_lake	max_daily_consumption_kpi	
PELL_3	false	24	16/06/2021 18:28	delta_lake	expected_annual_consumption_deviation_kpi	
PELL_4	false	24	16/06/2021 15:48	delta_lake	max_consumption_deviation_kpi	
WEATHERDATA	false	1	07/06/2021 17:49	enea-selfuser	weatherdata	

Figura 16 - Storage: interfaccia di gestione flussi

L'aggiunta di un flusso necessita di diversi passaggi, di seguito illustrati:

1. Raccolta del dato da una nuova sorgente

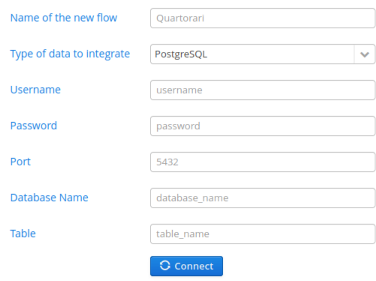
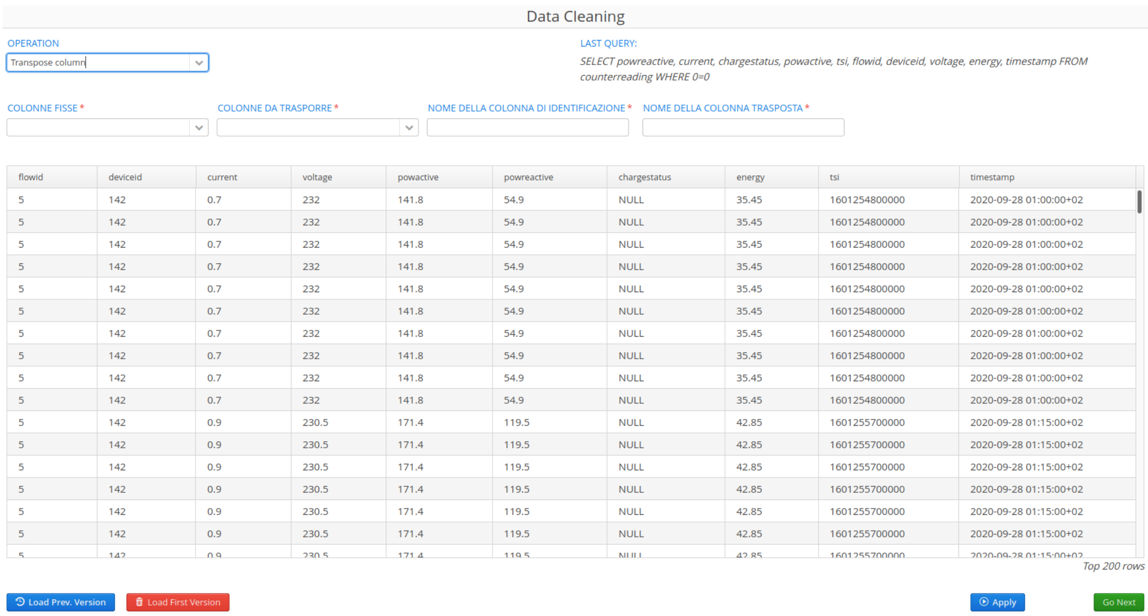


Figura 17 - Storage: interfaccia configurazione sorgente

2. Trasformazione del dato



Data Cleaning

OPERATION: Transpose column

LAST QUERY: *SELECT powreactive, current, chargestatus, powactive, tsi, flowid, deviceid, voltage, energy, timestamp FROM counterreading WHERE 0=0*

COLONNE FISSE * | COLONNE DA TRASPORRE * | NOME DELLA COLONNA DI IDENTIFICAZIONE * | NOME DELLA COLONNA TRASPOSTA *

flowid	deviceid	current	voltage	powactive	powreactive	chargestatus	energy	tsi	timestamp
5	142	0.7	232	141.8	54.9	NULL	35.45	1601254800000	2020-09-28 01:00:00+02
5	142	0.7	232	141.8	54.9	NULL	35.45	1601254800000	2020-09-28 01:00:00+02
5	142	0.7	232	141.8	54.9	NULL	35.45	1601254800000	2020-09-28 01:00:00+02
5	142	0.7	232	141.8	54.9	NULL	35.45	1601254800000	2020-09-28 01:00:00+02
5	142	0.7	232	141.8	54.9	NULL	35.45	1601254800000	2020-09-28 01:00:00+02
5	142	0.7	232	141.8	54.9	NULL	35.45	1601254800000	2020-09-28 01:00:00+02
5	142	0.7	232	141.8	54.9	NULL	35.45	1601254800000	2020-09-28 01:00:00+02
5	142	0.7	232	141.8	54.9	NULL	35.45	1601254800000	2020-09-28 01:00:00+02
5	142	0.7	232	141.8	54.9	NULL	35.45	1601254800000	2020-09-28 01:00:00+02
5	142	0.7	232	141.8	54.9	NULL	35.45	1601254800000	2020-09-28 01:00:00+02
5	142	0.9	230.5	171.4	119.5	NULL	42.85	1601255700000	2020-09-28 01:15:00+02
5	142	0.9	230.5	171.4	119.5	NULL	42.85	1601255700000	2020-09-28 01:15:00+02
5	142	0.9	230.5	171.4	119.5	NULL	42.85	1601255700000	2020-09-28 01:15:00+02
5	142	0.9	230.5	171.4	119.5	NULL	42.85	1601255700000	2020-09-28 01:15:00+02
5	142	0.9	230.5	171.4	119.5	NULL	42.85	1601255700000	2020-09-28 01:15:00+02
5	142	0.9	230.5	171.4	119.5	NULL	42.85	1601255700000	2020-09-28 01:15:00+02
5	142	0.9	230.5	171.4	119.5	NULL	42.85	1601255700000	2020-09-28 01:15:00+02

Top 200 rows

Load Prev. Version | Load First Version | Apply | Go Next

Figura 18 - - Storage: interfaccia di trasformazione del dato

3. Mappatura e Integrazione dati

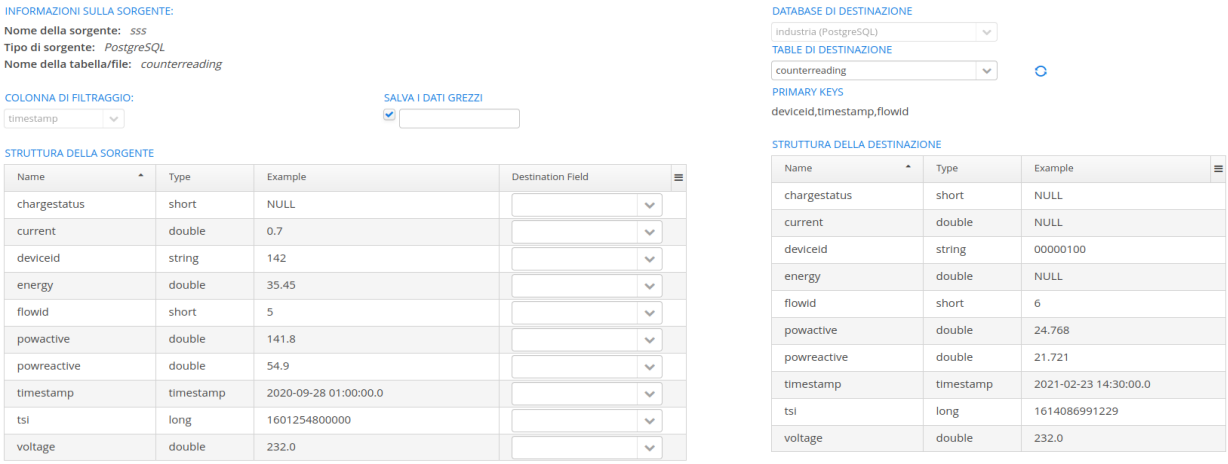


Figura 19 - Storage: interfaccia di mapping dati

- Configurazione del servizio di data retention

La Figura 20 mostra l'interfaccia per la visualizzazione e la modifica delle configurazioni del servizio di data retention.

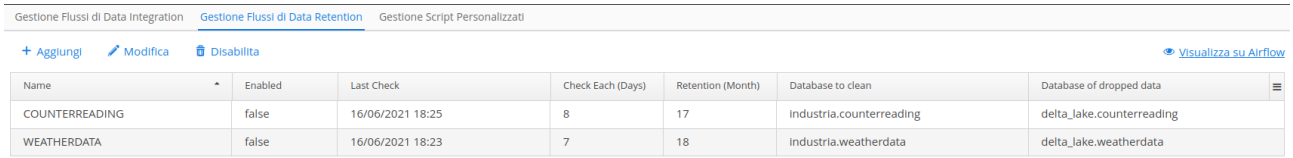


Figura 20 - Storage: interfaccia di gestione retention dati

Attraverso questa interfaccia è possibile modificare gli intervalli temporali (intervallo di aggiornamento e mesi di retention del dato), disabilitare una configurazione, visualizzare il DAG di Airflow e aggiungere una nuova configurazione.

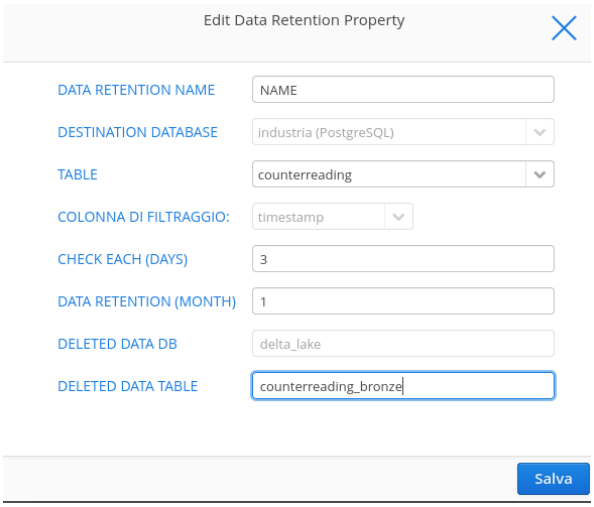
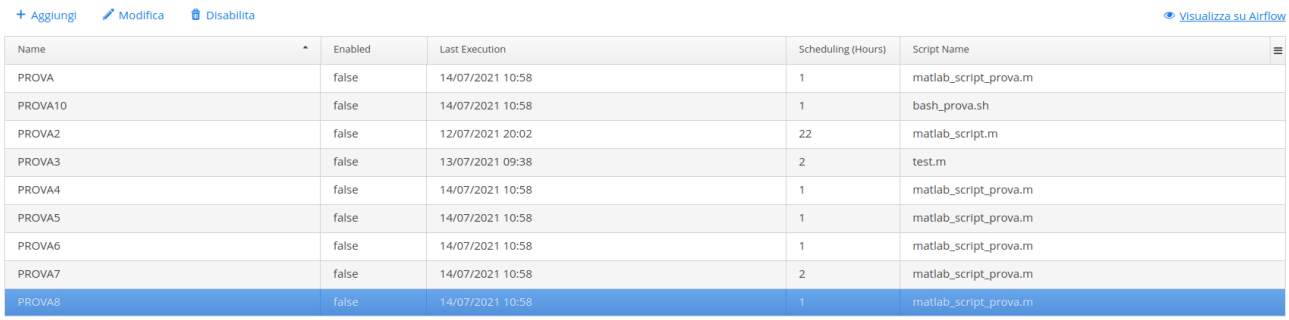


Figura 21 - Storage: interfaccia di amministrazione di una configurazione

- Configurazione del servizio di scheduling degli script customizzati

La Figura 21 mostra l'interfaccia per la visualizzazione e la modifica delle configurazioni del servizio scheduling degli scripts.



Name	Enabled	Last Execution	Scheduling (Hours)	Script Name
PROVA	false	14/07/2021 10:58	1	matlab_script_prova.m
PROVA10	false	14/07/2021 10:58	1	bash_prova.sh
PROVA2	false	12/07/2021 20:02	22	matlab_script.m
PROVA3	false	13/07/2021 09:38	2	test.m
PROVA4	false	14/07/2021 10:58	1	matlab_script_prova.m
PROVA5	false	14/07/2021 10:58	1	matlab_script_prova.m
PROVA6	false	14/07/2021 10:58	1	matlab_script_prova.m
PROVA7	false	14/07/2021 10:58	2	matlab_script_prova.m
PROVA8	false	14/07/2021 10:58	1	matlab_script_prova.m

Figura 22 - Storage: interfaccia di configurazione custom script

Attraverso questa interfaccia è possibile modificare l'intervallo temporale di scheduling, disabilitare una configurazione, visualizzare il DAG di Airflow e aggiungere una nuova configurazione.

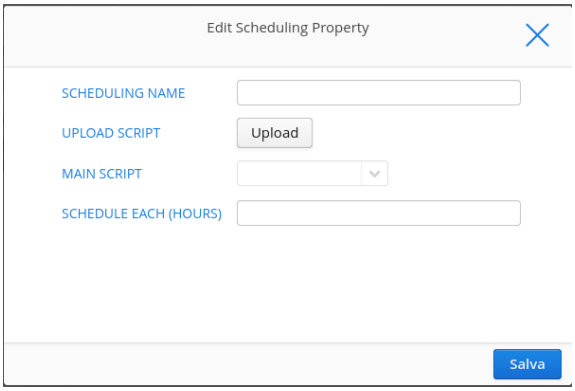


Figura 23 - Storage: interfaccia di modifica scheduling degli script

Nell'aggiunta di una configurazione è possibile inserire il nome della configurazione, l'intervallo di schedulazione e effettuare l'upload dello script e le sue librerie indicando il main script (metodo principale da cui avviare l'esecuzione dello script).

MOMIS non controlla in alcun modo lo script caricato, proprio per permettere la massima flessibilità richiesta dal progetto.

L'unico controllo che viene fatto è che il main sia in linguaggio Matlab (Octave) oppure bash.

4.4. Querying e Export

Per il servizio di interrogazione ed esportazione dei dati sono sviluppate nuove REST API che consentono un accesso sicuro ai dati integrati da MOMIS. Tutte le API utilizzano https con modalità POST. Di seguito vengono illustrati gli step necessari per la comunicazione tramite API Rest con autenticazione OAuth2:

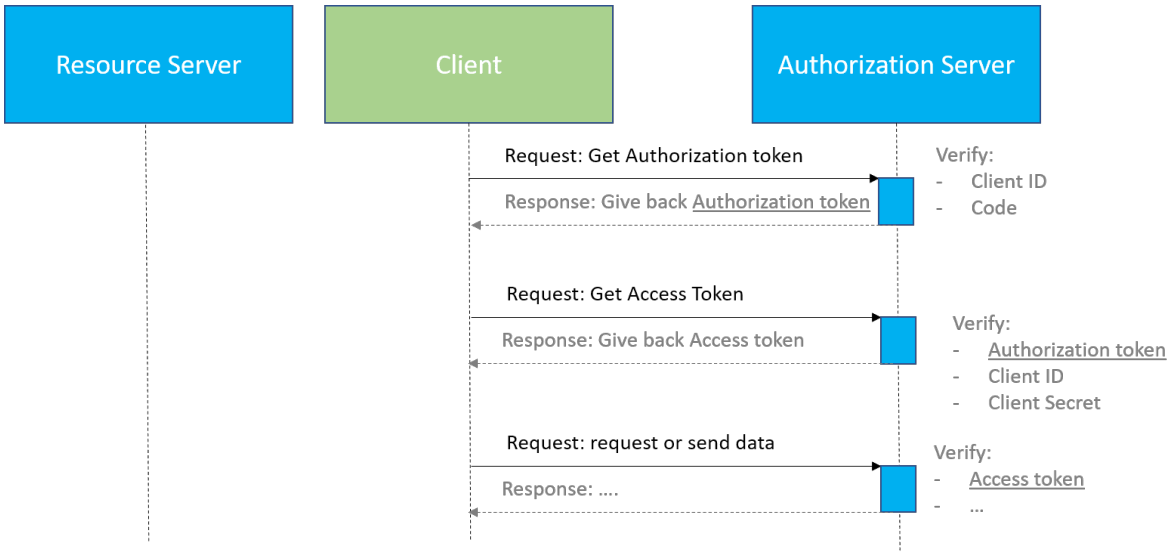


Figura 24 - Diagramma flusso di autenticazione delle REST API

Per poter utilizzare le API è necessario generare un token di sicurezza di durata limitata (3 ore di validità) e che dovrà essere utilizzato per tutte le chiamate.

La struttura della chiamata di generazione del token è il seguente:

URL: domain**/syncoauth/oauth2/token**

I request parameter sono i seguenti:

Campo	Tipologia Campo	Valore Campo
grant_type	string	client_credentials
client_id	string	Fornito da DataRiver
client_secret	string	Fornito da DataRiver
device_code	string	enea

I campi della risposta sono sono i seguenti:

Campo	Tipologia Campo	Valore Campo
access_token	string	access token
refresh_token	string	refresh token
expires_in	string	tempo di scadenza del token espresso in secondi

La risposta sarà del genere

{"access_token": "j38dj3d83jd9sk3osen28saj2j", "expires_in": “10800”, “refresh_token”: “kusa2ksna3d8aksw82sd2fr3t9y”}

E sarà associata ai seguenti codici di risposta

400 Bad Request: Il client ha inviato una richiesta non valida, ad esempio mancando il corpo della richiesta o il parametro richiesto (testo JSON non valido/parametri mancanti)

401 Unauthorized: L'autenticazione del client non è riuscita con il server (client_id errato o client_secret errato)

403 Forbidden: Client autenticato ma non dispone dell'autorizzazione per accedere alla risorsa richiesta (il codice errato / client_id non corrisponde)

500 Internal Server Error: Errore generico da notificare al server di DataRiver

Il token può anche essere rinfrescato per estenderne la validità, con la chiamata strutturata nel modo seguente:
URL: domain**/syncoauth/oauth2/token**

I request parameter sono i seguenti:

Campo	Tipologia Campo	Valore Campo
grant_type	string	refresh_token
client_id	string	Fornito da DataRiver
client_secret	string	Fornito da DataRiver
refresh_token	string	refresh token ottenuto con la chiamata al paragrafo 3.3

I campi della risposta sono sono i seguenti:

Campo	Tipologia Campo	Valore Campo

Campo	Tipologia Campo	Valore Campo
access_token	string	access token
refresh_token	string	refresh token
expires_in	string	tempo di scadenza del token espresso in secondi

La risposta sarà del genere

{"access_token": "j38dj3d83jd9sk3osen28saj2j", "expires_in": “10800”, “refresh_token”: “kusa2ksna3d8aksw82sd2fr3t9y”}

E sarà associata ai seguenti codici di risposta:

400 Bad Request: Il client ha inviato una richiesta non valida, ad esempio mancando il corpo della richiesta o il parametro richiesto (testo JSON non valido / parametri mancanti)

401 Unauthorized: L'autenticazione del client non è riuscita con il server (client_id errato o client_secret errato)

403 Forbidden: Client autenticato ma non dispone dell'autorizzazione per accedere alla risorsa richiesta (il codice errato / client_id non corrisponde)

500 Internal Server Error: Errore generico da notificare al server di DataRiver

4.4.1. API Esportazione in UrbanDataset

È stata implementata l’API di interrogare la tabella delle letture e di esportare i valori in formato UrbanDataSet-XML, UrdanDataSet-JSON, o CSV. Solo se il parametro access_token è valido, sarà possibile accedere ai dati e visualizzarli. Opzionalmente consente anche di validare l’UbanDataSet (JSON o XML) sulla base della configurazione dello schematron; in questo caso, e solo se la validazione fallisce, viene fornito un report in formato JSON contenente:

- l’esito della validazione (failed)
- lista degli schemi errors o schematron errors
- UrbanDataSet ottenuto

Solo se il parametro access_token è valido, sarà possibile accedere ai dati e visualizzarli.

URL: domain**/dashboard/exportservice/exportud**

REQUEST PARAMETERS

FIELD	FIELD TYPE	FIELD VALUE
access_token	string	Access token ritornato della API “GetAccessToken”

FIELD	FIELD TYPE	FIELD VALUE
format	string	Formato risposta, valori ammessi: - csv - json - xml
startdate	date	Data di inizio: formato aaaa-mm-gg
enddate	date	Data di inizio: formato aaaa-mm-gg
application_id	string	valore ammesso: “uds_extraction”
validate_uds	string	abilita la validazione dell’UD. Valori ammessi: - true - false

RESPONSE

Se format=JSON → Urban dataset in formato JSON

Se format=XML → Urban dataset in formato XML

Se format=CSV → Elenco delle letture in formato CSV

PS per questioni di leggibilità di questo documento non abbiamo allegato gli urban dataset

in caso di errori di validazione (JSON o XML), json composto da 3 campi:

FIELD	FIELD TYPE	FIELD VALUE
report_validation	string	Esito validazione. valore ritornato: failed
report	string	Urban dataset in formato JSON/XML
schematron_errors (oppure schema_errors)	string	lista degli schemi errors o schematron errors

esempio


```
{   "schematron_errors": [
            "[UD-008]- Il numero di proprieta' ...",
          .....
            "[UD-009]- La proprieta' Voltage non appartiene all'UrbanDataset."
```

],

"report_validation": "failed",

"report": **Urban dataset in formato JSON/XML**

}

Questa funzionalità è stata progettata per consentire la configurazione della API senza dover intervenire sul codice sorgente. A questo proposito di seguito vengono forniti i dettagli relativi alla struttura e configurazione di questa funzionalità.

4.4.2. Struttura Modulo di Esportazione in UrbanDataset

Il modulo di esportazione in UDS consente di configurare la struttura sia della query sui dati da esportare in formato UrbanDataset che i template dei formati UrbanDataset-XML e UrbanDataset-JSON e infine lo schematron per la validazione.

I file di configurazioni si trovano nelle cartelle:

/usr/home/momis/application_files/templates: contiene il file dello schematron

- UD-schematron.sch

/usr/home/momis/application_files/templates/json: contiene il file con il JSON Schema e il template JSON

- UD-schema.json
- Ud-test.json

/usr/home/momis/application_files/templates/xml: contiene il file con il XSD Schema e il template dell'urbanDataset in formato XML

- UD-schema.xsd
- UD-template.xml

Inoltre parte della configurazione è salvato nel database della Web Application

database ECD platform: **tabella dashboard_configuration**

la tabella dashboard_configuration ha la seguente struttura:

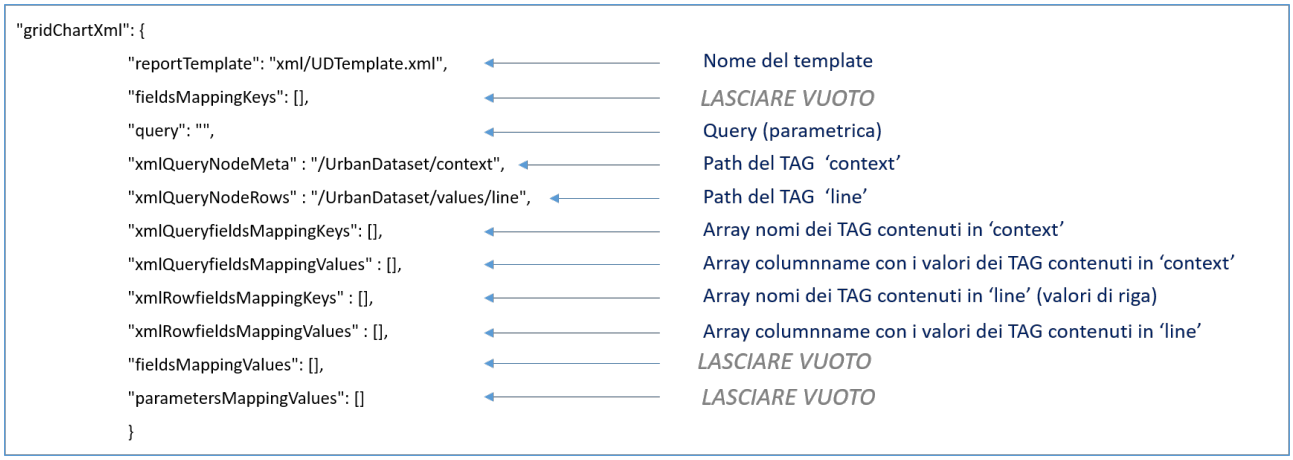
dashboard_configuration	
conf_id	INTEGER
conf_description	CHARACTER VARYING(255)
conf_json	CHARACTER VARYING
conf_version	INTEGER
conf_priority	INTEGER
conf_deleted	INTEGER
permission_read	INTEGER
author_id	INTEGER
creation_datetime	TIMESTAMP(6) WITHOUT TIME ZONE
permission_edit	INTEGER
permission_delete	INTEGER

definisce la configurazione della MOMIS Dashboard che è il engine dei servizi di analysis e export (ma non i servizi di queryning) è il campo “conf_json” a contenere la configurazione in formato JSON.

la struttura è la seguente:

```
{
        [...]
        "charts": [                          <-- Array dei 'chart' (grafici / griglie / esportazioni dati)
        {
            id": "grid_readings",            <-- Nome del chart delle esportazioni dell'UDS
            "connection":"enea-conn",        <-- ID della connessione utilizzata
            [..]
           "gridChartJson": {                <-- Configurazione dell'esportazione dell'UDS in formato JSON
             }
            "gridChartXml": {                <-- Configurazione dell'esportazione dell'UDS in formato JSON
             }
        [...]
        }
```

per la configurazione dell’esportazione dell’urbandataset in formato XML il nodo di riferimento è gridChartXml, che è strutturato nel seguente modo



NOTA: la query contiene la sottostringa ":condition" e a tempo di esecuzione della query viene compilato con le condizioni della query; per testare la correttezza della query sostituire la stringa con delle clausole where definite manualmente

ad esempio

```
"gridChartXml": {
                    "parametersMappingKeys": [],
                    "enabled": true,
                    "reportTemplate": "xml/UDTemplate.xml",
                    "fieldsMappingKeys": [],
                    "query": "SELECT flowname, podid AS PODID, linecurrent AS LineCurrent,
voltagephase AS PhaseVoltage, powerfactorphase AS PowerFactorPhase, reactivepowerphase AS
ReactivePower, timestamp_pk, reactiveenergy as ActiveEnergy, line_id, line_start_ts, line_end_ts,
line_format, line_latitude,  line_longitude,  line_height ,   query_id, query_scheme_id AS
MeterSchemeID, query_timeZone,  query_timestamp ,  query_language, query_format, query_latitude,
query_longitude, query_height  FROM counterreading_full WHERE :conditions ORDER BY timestamp_pk
ASC LIMIT 20",

                    "xmlQueryNodeMeta" : "/UrbanDataset/context",
                    "xmlQueryNodeRows" : "/UrbanDataset/values/line",
```

```
"xmlQueryfieldsMappingKeys" : [
                                        "[producer/id]",
                                        "[producer/schemeID]",
                                        "[timeZone]",
                                        "[timestamp]",
                                        "[language]",
                                        "[coordinates/format]",
                                        "[coordinates/latitude]",
                                        "[coordinates/longitude]",
                                        "[coordinates/height]"
                                        ],
"xmlQueryfieldsMappingValues" : [
                                        "query_id",
                                        "query_scheme_id",
                                        "query_timeZone",
                                        "query_timestamp",
                                        "query_language",
                                        "query_format",
                                        "query_latitude",
                                        "query_longitude",
                                        "query_height"
],
"xmlRowfieldsMappingKeys" : [
                                "[line_id]",
                                        "[schemeID]",

                                        "[period/start_ts]",
                                        "[period/end_ts]",
                                        "[coordinates/format]",
                                        "[coordinates/latitude]",
                                        "[coordinates/longitude]",
                                        "[coordinates/height]",
                                        "[activeenergy]",
                                        "[podid]",
                                        "[reactivernergy]",
                                        "[linecurrent]",

                                        "[reactivepowerphase]",
                                        "[phasevoltage]"
 ],
```

```
                "xmlRowfieldsMappingValues" : [
                                "line_id",
                                "query_scheme_id",
                                "line_start_ts",
                                "line_end_ts",
                                "line_format",
                                "line_latitude",
                                "line_longitude",
                                "line_height",
                                "ActiveEnergy",
                                "PODID",
                                "ReactiveEnergy",
                                "LineCurrent",
                                "ReactivePowerPhase",
                                "PhaseVoltage"
                 ],
                "fieldsMappingValues": [],
                "parametersMappingValues": []
        }
```

per la configurazione dell'esportazione dell'urbandataset in formato JSON il nodo di riferimento è gridChartJson, che è molto simile al nodo gridChartXml ed è strutturato nel seguente modo:

```
"gridChartJson": {
        "parametersMappingKeys": [],                          <-- LASCIARE VUOTO
        "enabled": true,
        "reportTemplate": "json/Ud-test.json",                <-- Nome del template
        "fieldsMappingKeys": [],                              <-- LASCIARE VUOTO
        "query": "",                                          <-- Query (parametrica)
        "jsonQueryNodeMeta" : "/UrbanDataset/context",        <-- Path del Nodo 'context'
        "jsonQueryNodeRows" : "/UrbanDataset/values/line",    <-- Path del Nodo 'line'
        "jsonQueryfieldsMappingKeys" : [],                    <-- Array nomi dei Nodi contenuti in 'context'
        "jsonQueryfieldsMappingValues" : [                    <-- Array columnname con i valori dei nodi contenuti in 'context'
        "jsonRowfieldsMappingKeys" : [],                      <-- Array nomi dei nodi contenuti in 'line' (valori di riga)
        "jsonRowfieldsMappingValues" : [],                    <-- Array columnname con i valori dei nodi contenuti in 'line'
         "jsonRowPropertiesValues" : []                       <-- Array columnname con i valori dei nodi contenuti in 'properties'
        "fieldsMappingValues": [],                            <-- LASCIARE VUOTO
        "parametersMappingValues": []                         <-- LASCIARE VUOTO
        }
```

ad esempio:

```
"gridChartJson": {
                        "parametersMappingKeys": [],
                        "enabled": true,
                        "reportTemplate": "json/Ud-test.json",
                        "fieldsMappingKeys": [],
                        "query": "SELECT flowname, podid AS PODID, linecurrent AS LineCurrent,
voltagephase AS PhaseVoltage, powerfactorphase AS PowerFactorPhase, reactivepowerphase AS
ReactivePower, timestamp_pk, reactiveenergy as ActiveEnergy, line_id, line_start_ts, line_end_ts,
line_format, line_latitude,  line_longitude,  line_height ,   query_id, query_scheme_id AS
MeterSchemeID, query_timeZone,  query_timestamp ,  query_language, query_format, query_latitude,
query_longitude, query_height  FROM counterreading_full WHERE :conditions ORDER BY timestamp_pk
ASC LIMIT 20",
                        "jsonQueryNodeMeta" : "/UrbanDataset/context",
                        "jsonQueryNodeRows" : "/UrbanDataset/values/line",
                        "jsonQueryfieldsMappingKeys" : [
                                                "producer/id,schemeID",
                                                "timeZone",
                                                "timestamp",
                                                "language",
                                                "coordinates/format,latitude,longitude,height"
                                                ],
                        "jsonQueryfieldsMappingValues" : [
                                                "query_id,query_scheme_id",
                                                "query_timeZone",
                                                "query_timestamp",
                                                "query_language",
                                                "query_format",
                                                “query_latitude”,
                                                “query_longitude”,
                                                “query_height"
                        ],
                        "jsonRowfieldsMappingKeys" : [
                                                "id",
                                                "period/start_ts,end_ts",
                                                "coordinates/format,latitude,longitude,height",
                                                "[property]/name,val"
                         ],
```

```
                    "jsonRowfieldsMappingValues" : [
                                        "line_id",
                                        "line_start_ts,line_end_ts",
                                        "line_format,line_latitude,line_longitude,line_height",
                                        "PROPERTIES_FILE_FIELD,PROPERTIES_FILE_VALUE"
                    ],
                    "jsonRowPropertiesValues"                                                    :
["SchemeID","ActiveEnergy","PODID","ReactiveEnergy",
"LineCurrent","ReactivePower","PhaseVoltage"],
                    "fieldsMappingValues": [],
                    "parametersMappingValues": []
                },
```


4.4.3. API Query Dinamiche

Infine è stata anche sviluppata l'API per la gestione delle query dinamiche
Consente di creare dinamicamente delle query SQL parametrizzate associate ad uno specifico ruolo per poter leggere i dati.

URL: domain**/retrieve/data/get**

I parametri della richiesta saranno vincolati solamente dal numero definito in tabella, non vi sarà alcun tipo di controllo ulteriore. Questa implementazione può essere soggetta ad attacchi di SQL Injection e quindi causa di un fallimento durante un Penetration Test. Il passaggio di parametri liberi e non opportunamente vincolari costituisce un possibile punto di attacco malevolo.

Nonostante questa considerazione, procediamo come richiesto da ENEA.

La lista dei parametri ammessi è dunque la seguente:

Campo	Tipologia Campo	Valore Campo
access_token	String	Access token ritornato della API "GetAccessToken"
application_id	String	ecdp_app
query_code	string	codice della query
p1	String	parametro 1
p2	String	parametro 2
..	...	...
pN	String	parametro N-esimo

La risposta consiste nel JSON contenente i dati estratti dalla query

La risposta alla chiamata sarà associata ai seguenti codici di risposta:

400 Bad Request: Il client ha inviato una richiesta non valida, ad esempio mancando il corpo della richiesta o il parametro richiesto (testo JSON non valido/parametri mancanti)

401 Unauthorized: il client ha fallito l'autenticazione con il server (wrong access_token)

403 Forbidden: il client si è autenticato ma non ha il permesso di accedere alla risorsa richiesta (access_token expired)

500 Internal Server Error: Errore generico da notificare al server di DataRiver

Per le query dinamiche è stata implementata una logica di controllo che verifica che l'utente abbia accesso alla query con i permessi di lettura che il numero di parametri inseriti sia corretto e che la lunghezza dei parametri inseriti sia nel range previsto.

4.5. Source Management Dashboard

La Source Management Dashboard è uno strumento in grado di visualizzare e monitorare i dati senza intervenire sulle sorgenti e mantenendo i vincoli di visibilità per ciascun tipo di utente.
La Source Management Dashboard consiste in interfacce grafiche di gestione delle sorgenti dati che ne consentono il monitoraggio: ultimo aggiornamento, stato della sorgente all'ultima richiesta dei dati, retention sullo storage relazionale, problematiche riscontrate durante il processo di integrazione di nuovi dati.

Questi indicatori possono essere utilizzati per gestire i flussi di dati andando a correggere l'integrazione ove necessario.

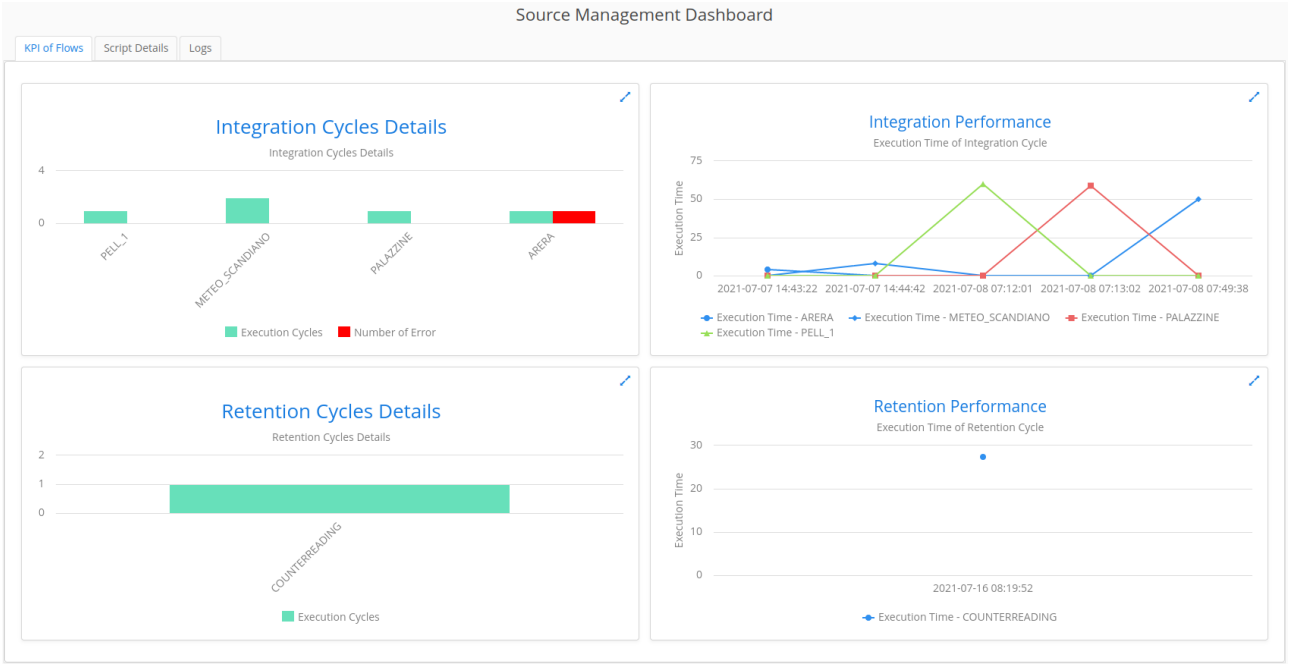


Figura 25 - UI della Source Management Dashboard

La Source Management Dashboard è composta da un primo set di quattro grafici realizzati per avere un'idea dell'andamento generale dei job di integrazione dati e di retention dei dati sulla sorgente relazionale.

- Integration Cycles Details e Retention Cycles Details

Consentono il monitoraggio dei flussi mostrando, per ogni flusso configurato, il numero di cicli di aggiornamento effettuati con successo e il numero di cicli falliti.

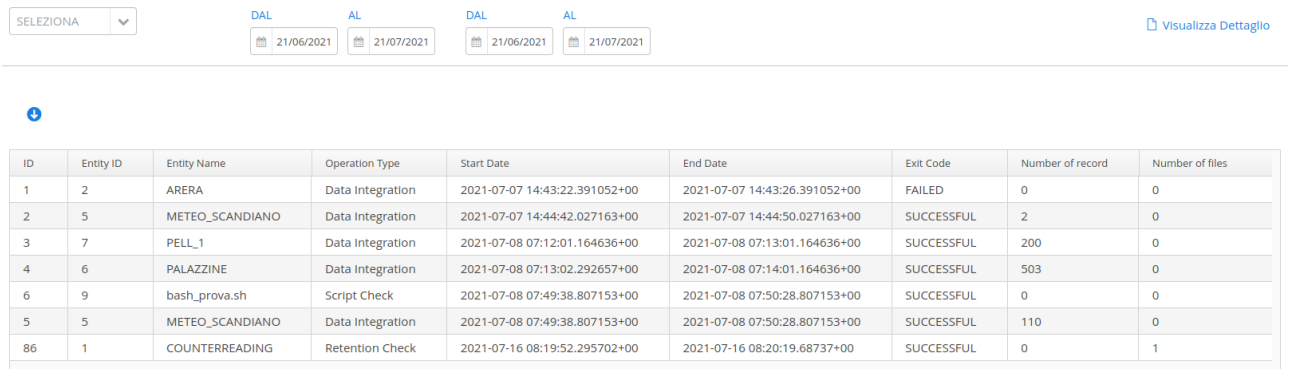
- Integration Performance e Retention Performance

Per ogni flusso definito, mostrano il tempo impiegato per integrare nuovi dati dalla sorgente (Integration Performance) o il tempo per spostare i dati che hanno oltrepassato il periodo di retention (Retention Performance).

- Script Customizzati

Il servizio di scheduling di script customizzati prevede anch'esso due dashboard descrittive dei cicli di esecuzione svolti e del tempo di esecuzione impiegato.
Nella prima dashboard, per ogni script configurato, viene mostrato il numero di cicli di esecuzione effettuati con successo e il numero di cicli falliti.
Nella seconda dashboard, per ogni script configurato, viene mostrato il tempo impiegato concludere l'esecuzione.

- Monitoraggio job di Integrazione, retention ed esecuzione script



ID	Entity ID	Entity Name	Operation Type	Start Date	End Date	Exit Code	Number of record	Number of files
1	2	ARERA	Data Integration	2021-07-07 14:43:22.391052+00	2021-07-07 14:43:26.391052+00	FAILED	0	0
2	5	METEO_SCANDIANO	Data Integration	2021-07-07 14:44:42.027163+00	2021-07-07 14:44:50.027163+00	SUCCESSFUL	2	0
3	7	PELL_1	Data Integration	2021-07-08 07:12:01.164636+00	2021-07-08 07:13:01.164636+00	SUCCESSFUL	200	0
4	6	PALAZZINE	Data Integration	2021-07-08 07:13:02.292657+00	2021-07-08 07:14:01.164636+00	SUCCESSFUL	503	0
6	9	bash_prova.sh	Script Check	2021-07-08 07:49:38.807153+00	2021-07-08 07:50:28.807153+00	SUCCESSFUL	0	0
5	5	METEO_SCANDIANO	Data Integration	2021-07-08 07:49:38.807153+00	2021-07-08 07:50:28.807153+00	SUCCESSFUL	110	0
86	1	COUNTERREADING	Retention Check	2021-07-16 08:19:52.295702+00	2021-07-16 08:20:19.68737+00	SUCCESSFUL	0	1

Figura 26 - UI interfaccia di monitoraggio job

Sulla Source Management Dashboard sono state anche realizzate le intefacce di monitoraggio con il dettaglio relativo ai cicli di esecuzione: nome del flusso o script, tipo di operazione effettuata (Data Integration, Retention Check o Script Check) il timestamp di inizio e di fine, il codice di uscita (FAILED o SUCCESSFUL), il numero di record coinvolti e, nel caso di integrazione con i files, numero di files integrati.
In questa sezione è disponibile anche un servizio di estrazione dati, filtraggio dei risultati e visualizzazione del dettaglio. Tramite il pulsante in alto a destra è possibile visualizzare il dettaglio dei log registrati durante tutto il ciclo

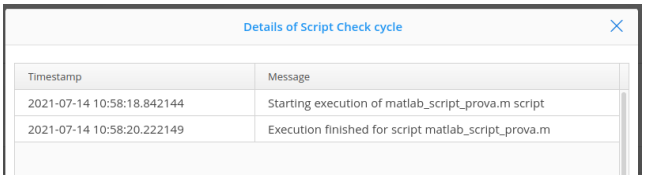


Figura 27 – Dettaglio esecuzione Job

5. Struttura codice ECD Platform

La Energy Community Data Platform è stata sviluppata per avere una struttura modulare e facilitare il lavoro di sviluppo anche in ottica di realizzazione di future estensioni.
per dare una struttura modulare al codice è stato utilizzato Apache Maven, uno strumento di automazione di costruzione per progetti Java. Maven affronta due aspetti del software costruzione: In primo luogo, si descrive come il software è costruito, e la seconda, descrive le sue dipendenze. Un file XML descrive il progetto di software in fase di costruzione, le sue dipendenze da altri moduli esterni e componenti, l'ordine di generazione, le directory ei plug-in necessari. Viene fornito con 'goals' predefiniti per l'esecuzione di determinati compiti ben definiti, come la compilazione di codice e la generazione dei file di archivio java (JAR e WAR). Maven scarica dinamicamente librerie Java e Maven plug-in da uno o più repository come il Maven Central Repository, e li memorizza in una cache locale.

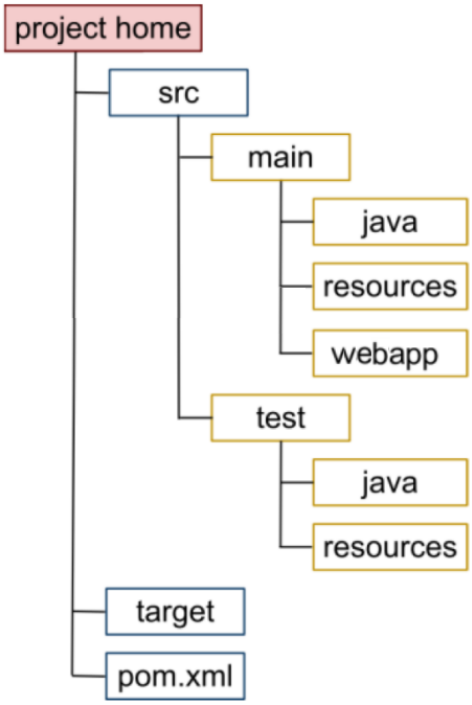
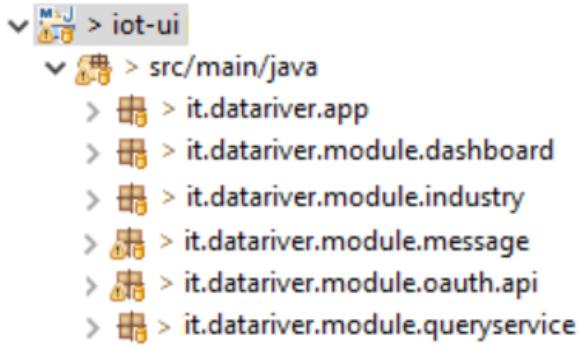


Figura 28 – Struttura generica di un progetto Maven

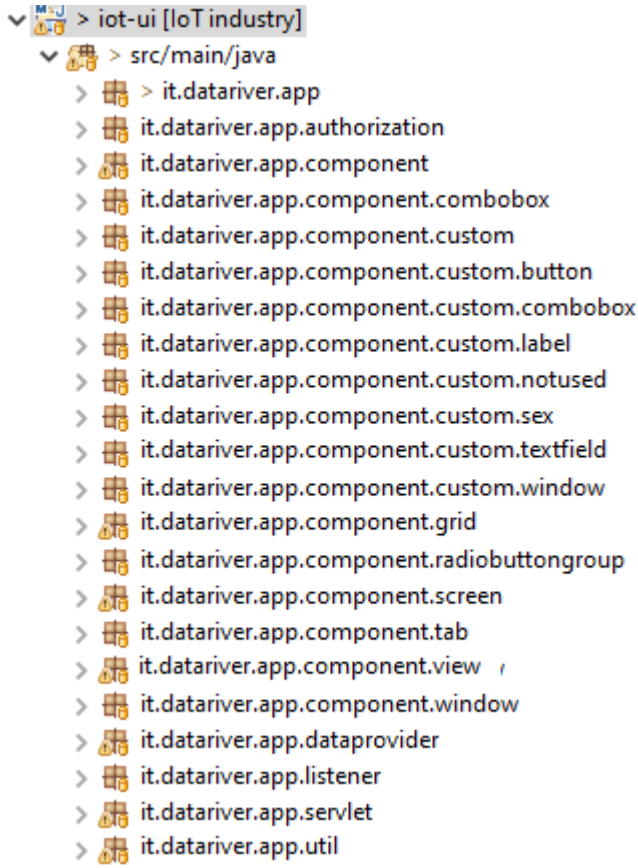
La struttura del progetto è definita da Maven per cui si possono distinguere le directory:

- **src/main/ java:** contiene i sorgenti java della piattaforma;
- **src/main/resources**: contiene le risorse dell'applicazione;
- **src/main/webapp**: contiene le risorse specifiche della 'applicazione web (elementi grafici e file configurazione web);
- **pom.xml**: contiene informazioni sul progetto, le informazioni sulle librerie utilizzate, dettagli di configurazione utilizzati da Maven per creare il progetto.

I sorgenti, sono organizzati i package in base alla loro funzione implementata dai metodi delle classi Java in essi contenute. i package realizzati nella piattaforma sono mostrati nella figura sottostante:



- **it.datariver.app:** contiene le classi di base della piattaforma per la creazione degli elementi grafici e la gestione della web servlet e delle sessioni. Nel package è presente la classe ***it.datariver.app.AppUI*** che costituisce l'entry point della web application , e ovviamente anche della documentazione (JavaDoc) relativa al codice sorgente.
  il package ovviamente è organizzato in sottopackage al fine di migliorare l'usabilità del codice e favorirne la leggibilità:

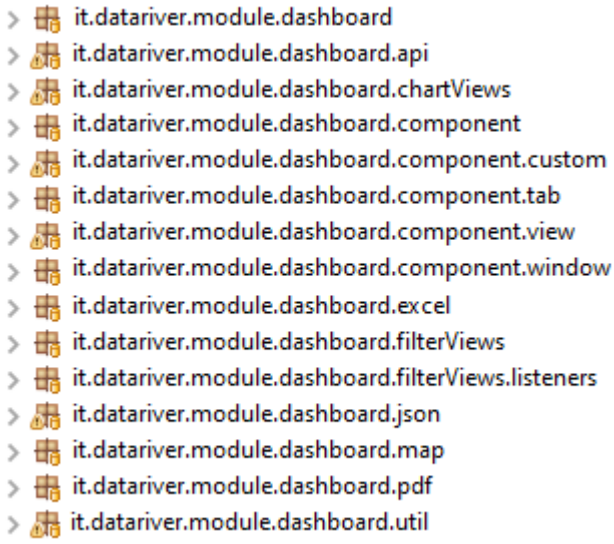


  nel sottopackage ***Component*** sono presenti tutti gli elementi grafici della web application (pulsanti, tabelle, label, combobox, finestre di dialogo ...), principalmente realizzate estendendo la libreria com.vaadin.ui;

  nel sottopackage ***Authorization*** sono presenti classi e interfacce per la gestione delle politiche di accesso alla piattaforma;

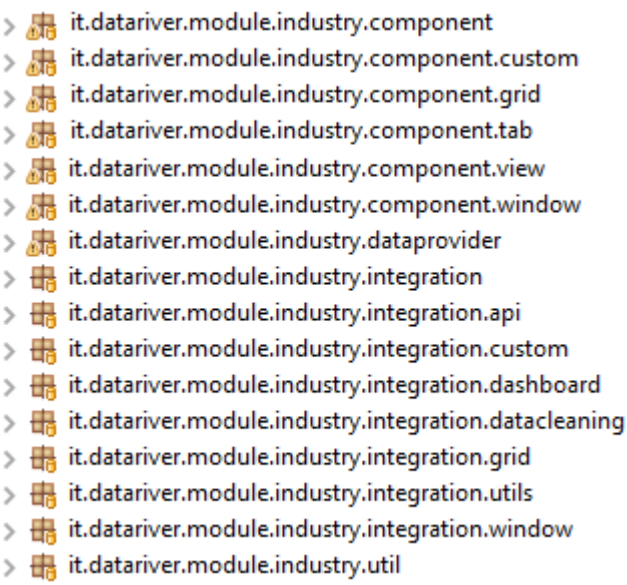
il sottopackage **Dataprovider** contiene le classi che definiscono gli oggetti per la gestione dei dati da visualizzare tramite gli oggetti grafici definiti nel sottopackage component
il sottopackage **Listener** contiene le classi per la gestione degli eventi scatenati dall'interazione con gli elementi grafici (listener appunto)
il sottopackage **Servlet** contiene le classi per la gestione della httpServlet
il sottopackage **Util** contiene le classi per la definizione di metodi ausiliari comuni per il progetto

- **it.datariver.module.dashboard:** contiene le classi specifiche del modulo di data analysis e di esportazione dati (anche formato Urban Dataset-XML e Urban Dataset-JSON), e che si basa sull'engine MOMIS Dashboard ovvero la soluzione di DataRiver di data Analytics:



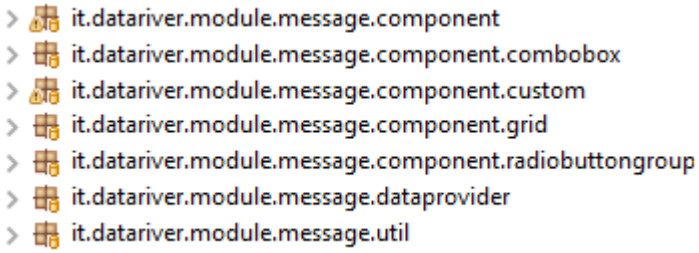
i principali sottopackage sono:
Component: contiene le classi per la creazione degli elementi grafici specifici per la dataAnalytics
ChartViews, Map e Filterviews: contiene classi per la generazione di grafici mappe e filtri
API: contiene le classi che definiscono la logica per l'estrazione dei dati in formato Urban Dataset

- **it.datariver.module.industry:** contiene le classi utilizzate sia dal modulo di data ingestion e Data Cleaning e Data Integration:



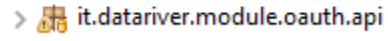
i principali sottopackage sono:
Component: contiene le classi per la creazione degli elementi grafici specifici dei moduli di data ingestion e di data integration
Integration: contiene le classi per la definizione delle logiche per la data integration

- **it.datariver.module.message:** contiene le classi utilizzate per la messaggistica della web application:

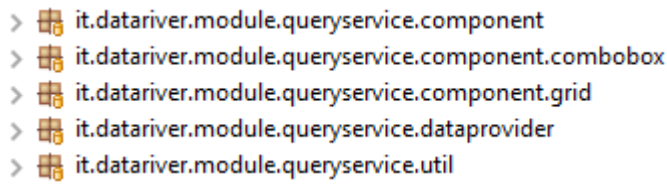


in questo caso, dato che ha solo i sottopackage comuni (**Component, Dataprovider e Util),** non vengono ripetuti i dettagli già illustrati in precedenza.

- **it.datariver.module.oauth:** contiene le classi utilizzate per realizzare le logiche di sicurezza secondo il protocollo OAuth2 utilizzato dalle Rest API dell'applicazione:



- **it.datariver.module.queryservice:** contiene le classi utilizzate nel modulo di querying e implementa la logica delle query personalizzate.



anche in questo caso sono presenti i sottopackage comuni **Component, Dataprovider e Util**.

La piattaforma della piattaforma è composta da un prodotto principale (MOMIS) e altri sottoprodotti anche di terze parti come illustrato nella figura sottostante con anche le relative porte.

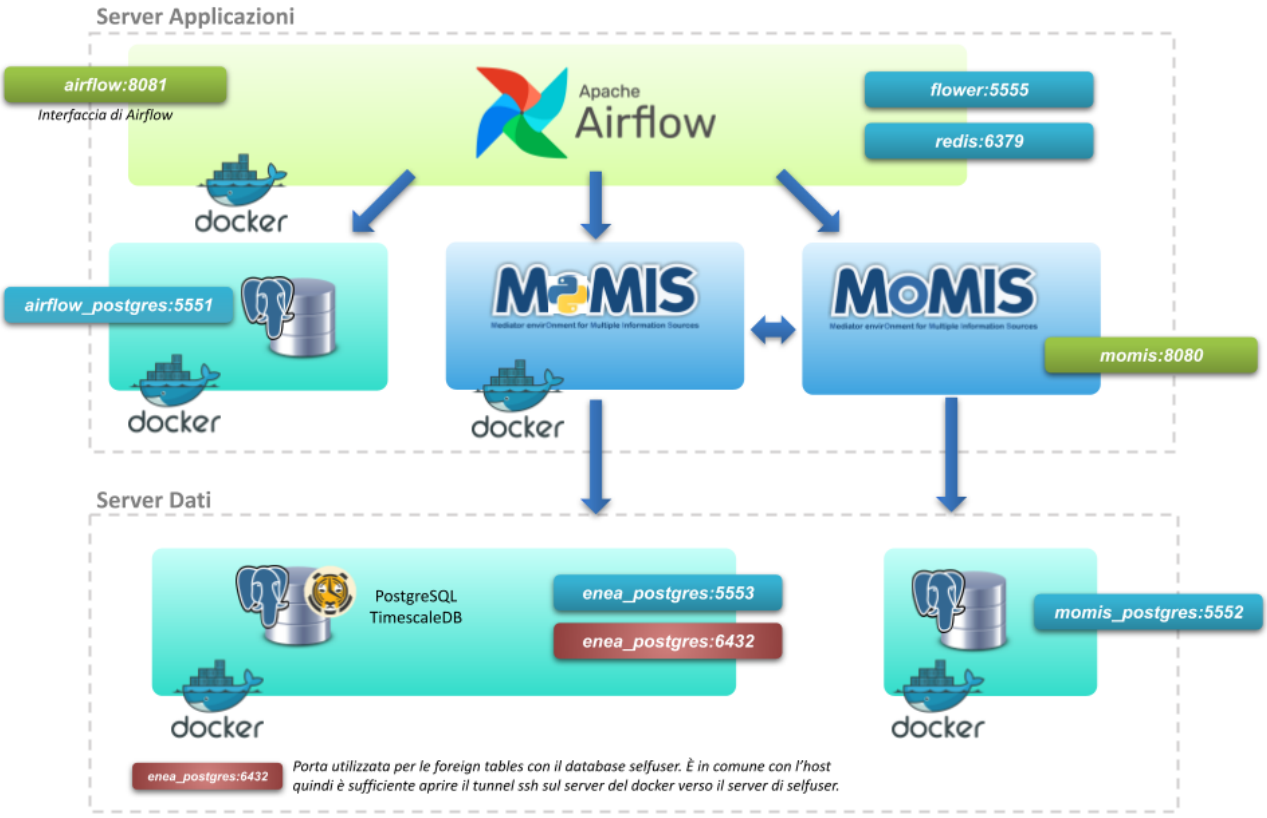


Figura 29 – Mappatura porte piattaforma ECD Platform