UNIVERSITÀ DEGLI STUDI
DI GENOVA

DIPARTIMENTO DI INGEGNERIA NAVALE,
ELETTRICA, ELETTRONICA E DELLE
TELECOMUNICAZIONI

Tesi di Laurea Magistrale in Ingegneria Elettronica

Marzo 2022

Progetto e implementazione di una API per la gestione di dataset in ambito IoT

(Design and development of an API for IoT datasets management)

Candidato: Matteo Fresta

Relatori: Prof. Riccardo Berta, Prof. Francesco Bellotti

Correlatori: Dottorandi Alessio Capello, Marianna Cossu, Luca Lazzaroni

Sommario

Il presente elaborato è stato redatto con l'obiettivo di illustrare la progettazione e l'implementazione di una API[1] (*Application Programming Interface*) per la gestione di dataset in ambito *Internet of Thing* (IoT).

L'interfaccia estende il framework IoT *Measurify*[2] (Una API astratta orientata alla gestione delle misure e basata sul *cloud*) per gestire la comunicazione con un utente *data analyst*, che utilizza dati IoT per realizzare vari tipi di modelli e analisi, specie nell'ambito dell'intelligenza artificiale e del machine learning.

La tesi ha implementato nuove funzionalità per un'efficiente gestione (caricamento, scaricamento, versionamento) di grandi quantità di dati, aumentando le capacità e le prestazioni di Measurify nella gestione di *Big Data*.

Per incrementare l'usabilità della piattaforma, è stata creata una libreria *Python* che espone allo sviluppatore le principali funzionalità della API in maniera efficiente ed intuitiva.

Il progetto della tesi ha così affrontato ed armonizzato due realtà estremamente attuali ed interconnesse: da una parte la diffusione esponenziale di oggetti fisici della nostra vita quotidiana estesi alle potenzialità del web ed interconnessi tra loro (IoT), dall'altra la gestione dei *Big Data,* ossia grandi volumi di dati generati dai dispositivi in rete, che vengono analizzati per lo sviluppo di intelligenza artificiale e la creazione di modelli di analisi, settori in continua espansione e sempre più rilevanti a livello economico globale.

Indice

1. Introduzione

La presente tesi ha come oggetto lo sviluppo di una API per la gestione di dataset in ambito IoT[3], aggiungendo funzionalità a *Measurify*[2], un progetto ideato, sviluppato e mantenuto da Elios Lab dell'Università degli Studi di Genova.

Questo progetto può trovare applicazione nei settori dell'*IoT*, dell'industria 4.0 e dell'automazione, i quali, in un questo momento di sviluppo tecnologico, costituiscono gli elementi chiave per la crescita economica di un paese.

Con termine IoT (*Internet of Things*) si vuole indicare la possibilità di estensione della rete Internet a tutti gli oggetti in grado di supportare le tecnologie necessarie: l'idea di base è che tutte le *things* possano comunicare tra loro attraverso la rete Internet, scambiando dati e informazioni e agendo in base a essi.

Gli oggetti diventano di conseguenza smart, capaci di interagire con la realtà in maniera autonoma e prendere delle decisioni a seguito di determinati input; un elevato numero di oggetti connessi tra loro necessita di uno scambio di informazioni in tempo reale sempre più ingente e per questo si parla di *Big Data* per indicare il gran numero di informazioni che vengono inviate e processate giornalmente. In questo mondo, la mole dei dati è dell'ordine degli *zettabyte*, ovvero miliardi di *terabyte*: per poter elaborare tutto ciò, sono necessarie tecnologie all'avanguardia che si affidano su calcolo massivo in parallelo eseguito anche su centinaia di server.

Altro aspetto fondamentale da considerare è lo scambio di informazioni e per questo si parlerà anche delle caratteristiche della rete 5G in continuo sviluppo nel contesto contemporaneo e che porterà sviluppi molto importanti nel mondo IoT.

In un futuro prossimo, la messa in funzione completa della rete 5G aprirà le porte allo scambio di grandi quantità di dati a velocità sempre più elevata e con un numero sempre più grande di dispositivi connessi.

Questa tesi terrà in considerazione gli aspetti sopraelencati e gestirà la comunicazione dei dati proveniente da sensori o da dataset fino alla memorizzazione su un database. Successivamente verranno affrontate le richieste di analisi, inviando i dati in vari formati. Possiamo riassumere l'ambiente in cui la seguente tesi è inserita in questo modo:

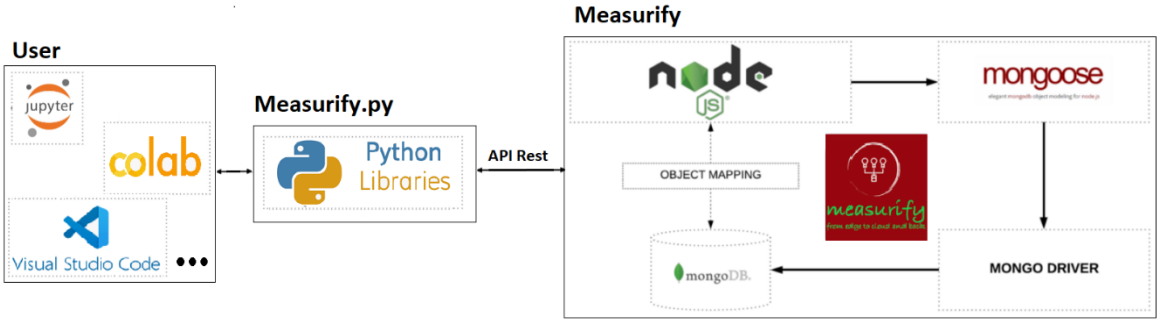

Figura 1: Architettura del sistema

Possiamo dividere l'architettura della tesi in tre sottoinsiemi: la prima è quella della API Framework Measurify[2], per poi collegarsi tramite le API Rest[4] alla libreria Python Measurify.py che metterà in comunicazione all'utente finale che potrà utilizzare diversi ambienti di sviluppo in Python.

Nello specifico all'interno di Measurify il punto di partenza è MongoDB[5] ossia il database utilizzato da Elios Lab dell'Università di Genova per memorizzare i dati sul cloud: MongoDB è un database di documenti NoSQL senza schema: è dunque possibile memorizzare documenti JSON, e la struttura di questi documenti può variare in quanto non è forzata come i database relazionali SQL. Questo è uno dei vantaggi dell'uso del NoSQL in quanto accelera lo sviluppo delle applicazioni e riduce la complessità delle implementazioni.

Per connettere la API Measurify, che si appoggia a Node.js[6], al server MongoDB tramite i Mongo Driver si utilizza Moongoose[7], libreria *Object Data Modeling* (ODM) per MongoDB e Node.js: questo gestisce le relazioni tra i dati, fornisce la

convalida dello schema ed è usato per interpretare gli oggetti nel codice con la loro rappresentazione in MongoDB.

La parte centrale del sistema è dunque **Node.js,** un ambiente *runtime* JavaScript *open-source*, multipiattaforma che esegue il codice JavaScript al di fuori di un browser web; al suo interno, viene eseguito il codice dell'API che tiene conto delle *get* e *post* ricevute e fornisce tramite delle rotte le informazioni richieste.

Per interconnettere in modo semplice e efficiente la API con l'ambiente di sviluppo in Python come per esempio i *Notebook documents*, è stata sviluppata una *Python library* chiamata Measurify.py utile a fornire tutte le chiamate necessarie per caricare o ricevere dataset di valori senza entrare nello specifico delle rotte necessarie alla API.
Questa libreria è stata ideata per semplificare i passaggi così che gli utenti possano accedere ai dati anche senza comprendere a fondo l'architettura di Measurify.

Infine, nello schema sono presenti editor di codice sorgente come Visual Studio code e i *Notebook documents* come Jupyter o Colab ossia documenti che contengono sia codici che elementi di testo come paragrafi, equazioni, figure, link. Questi, pertanto, sono documenti leggibili dall'uomo poiché contengono commenti sull'analisi in esame e risultati in formato di figure o tabelle.

Il principale flusso di informazioni consiste nello scambio di misurazioni singole, in gruppo o all'interno di un dataset. Nel caso di dati tabulari, ogni colonna di una tabella rappresenta una particolare variabile, così come ogni riga corrisponde a una misurazione dell'insieme di dati in questione.

La tesi è articolata in cinque capitoli:

- Nel primo viene descritto il contesto in cui ci troviamo al momento della stesura di questo elaborato analizzando lo sviluppo delle tecnologie fino ad arrivare allo stato dell'arte del progetto in questione.
- Nel secondo viene descritto approfonditamente il sistema in esame e tutte le loro componenti.
- Nel terzo vengono elencati e discussi gli approcci utilizzati per affrontare i diversi problemi ed è analizzato lo sviluppo del codice che ha portato alla versione definitiva caricata sulla API e alla versione finale della libreria Python.
- Nel quarto vengono descritti gli esperimenti effettuati su dataset reali per la verifica e ottimizzazione del codice e valutarne l'effettiva utilità.
- Infine, nel quinto e ultimo paragrafo sono esposte alcune considerazioni finali e il possibile sviluppo futuro.

2. Contesto di sviluppo

2.1 Internet of Things

Il termine "Internet of Things"[3](acronimo di **IoT**) è stato coniato da Kevin Ashton nel 1999; in quel periodo Ashton considerava l'identificazione a radiofrequenza (RFID) come essenziale per l'Internet of Things. Il tema principale dell'IoT è quello di incorporare ricetrasmettitori mobili a corto raggio in vari gadget e oggetti di uso quotidiano per consentire nuove forme di comunicazione tra le persone e le *cose.*

Cisco Systems ha stimato che l'IoT ha avuto origine tra il 2008 e il 2009; a conferma di ciò, si stima che il rapporto cose/persone sia cresciuto da 0,08 nel 2003 a 1,84 nel 2010.

Si valuta che entro il 2030 i dispositivi connessi ad Internet saranno circa 125 miliardi di unità, quadruplicando il numero del 2021 di circa 31 miliardi[8]. Il *McKinsey Global Institute*[9] prevede che il potenziale valore economico che l'IoT genererà nei prossimi anni sarà in forte crescita: si ritiene infatti che si registrerà un valore complessivo a livello globale compreso tra **5,5 e 12,6 trilioni di dollari** grazie all'applicazione dell'Internet delle Cose in diversi ambiti tra cui industria, sanità, uffici, città, cantieri, commercio e infine, case.

La possibilità di nuovi mercati e di ottimizzazione dei processi tecnologici ha portato a un grande sviluppo di questo settore da parte delle aziende che stanno investendo sempre più risorse in queste tecnologie. La stessa ricerca e sviluppo basata sull'Industria 4.0 e l'Industria IoT ha portato alla crescita del valore economico di quest' ultimo campo: i vantaggi partono ovviamente dai bassi costi delle tecnologie utilizzate e all'ampliamento continuo dell'applicazione dei **sensori** nei differenti campi.

In parallelo, possiamo parlare dello sviluppo della rete **5g** che garantirà una copertura capillare del territorio e scambio di dati ad elevate prestazioni.

Sotto il nome di **Internet of Things** vengono raccolti tutti gli oggetti ritenuti *smart*: non si parla soltanto di computer o smartphone, ma soprattutto degli oggetti che ci circondano all'interno delle nostre case, al lavoro, nelle città e nella vita di tutti i giorni.

I principali vantaggi di questa tecnologia sono:

- Accesso alla tecnologia dei sensori a **basso costo** e **bassa potenza**: sensori economici e affidabili stanno rendendo la tecnologia IoT possibile per più produttori.
- **Connettività**. Lo sviluppo di una serie di protocolli di rete per internet ha facilitato il collegamento di sensori al cloud e ad altre "cose" per un efficiente trasferimento dei dati.
- **Piattaforme di cloud computing**. L'aumento della disponibilità di piattaforme cloud permette sia alle aziende che ai consumatori di accedere all'infrastruttura di cui hanno bisogno e consente la scalabilità dei processi senza dover effettivamente comprare e gestire tutte le tecnologie computazionali e di storage necessarie.
- **Apprendimento automatico e analisi**. Con i progressi nel machine learning e nell'analisi statistica, insieme all'accesso a ricche quantità di dati memorizzati nel cloud, le aziende possono raccogliere informazioni e previsioni più velocemente e facilmente.
- **Intelligenza artificiale conversazionale (AI).** I progressi nelle reti neurali hanno portato l'elaborazione del linguaggio naturale ai dispositivi IoT (come gli assistenti personali digitali Alexa, Cortana e Siri) e li hanno resi piacevoli, accessibili e utilizzabili per l'uso domestico.

L’economicità delle componenti elettroniche ha portato a una vasta diffusione in tutto il mondo di elementi IoT, espandendosi in nuovi settori in cui è necessario analizzare la moltitudine di dati.

2.2 Domini applicativi

I principali campi di applicazione sono:

1. Smart City
2. Domotica
3. Robotica
4. Avionica
5. Industria automobilistica
6. Industria biomedicale
7. Difesa
8. Agricoltura

2.2.1 Smart City

Il concetto di "*smart city*" (ossia "città intelligente") introduce la progettazione di città tecnologicamente avanzate ai fini di migliorare la sostenibilità e la vivibilità umana.

La scelta di una rete IoT risulta essere quella più economica essendo i componenti di sensoristica non troppo costosi e ciò porta a una riduzione degli sprechi energetici. In una città, gli strumenti IoT posso essere utilizzati per il controllo del territorio e il monitoraggio ambientale oltre a essere anche utilizzati per il turismo come percorsi culturali o punti informativi.

Come vedremo nel capitolo delle sperimentazioni, a partire da dati ottenuti da sensori reali, verranno presi in considerazioni misurazioni di agenti inquinanti

dell'aria acquisiti da centraline sparse per la città di Bologna, che monitorano in tempo reale la concentrazione di fumi. Questi valori vengono utilizzati dal Comune al fine di imporre restrizioni in caso di peggioramento delle concentrazioni di sostanze nocive nell'aria con lo scopo di tutelare la salute dei cittadini.

Altri ambiti applicativi in città sono lo sviluppo di semafori intelligenti che gestiscono la rete stradale, così come un sistema di sorveglianza e di controllo della velocità in strada: questi sono tutti apparati "smart" che tramite appositi sensori gestiscono la circolazione connettendosi costantemente a internet per ricevere e inviare informazioni.

2.2.2 Domotica

Negli ultimi anni, il settore della domotica è stato quello in cui l'IoT ha avuto una diffusione più marcata: grazie alla connessione di dispositivi alla rete Internet, sempre più elementi in una casa sono stati resi "*smart*". Per la gestione di tali apparati sono stati sviluppati assistenti personali intelligenti come Alexa e Google Home che vengono definiti "*Smart Speaker*" ossia in grado di recepire funzioni vocali e rispondere alle nostre domande. Tramite questi accessori è possibile comandare gli altri oggetti "*smart*" installati in varie parti della casa come accendere luci o gestire altri elettrodomestici.
Il mondo degli elettrodomestici si sta avvicinando sempre di più al mondo smart e le possibilità di controllo da parte dell'utente tramite interfacce basate su smartphone sono sempre più diffuse.

2.2.3 Robotica

Nell'industria della robotica si sta sviluppando il concetto di "Internet of Robotic Things"[10] dove la differenza principale con l'Internet of Things come lo conosciamo è che i robot, operano attivamente nel mondo fisico. In altre parole, il dispositivo intelligente "sa" dove si trova e agisce di conseguenza.

ABI Research ha definito l'Internet of Robotic Things o IoRT, nel 2014 come:

"....il concetto di Internet of Robotic Things (IoRT), dove i dispositivi intelligenti possono monitorare gli eventi, fondere i dati dei sensori da una varietà di fonti, utilizzare l'intelligenza locale e distribuita per determinare una migliore linea d'azione, e quindi agire per controllare o manipolare gli oggetti il mondo fisico, e in alcuni casi mentre si muove fisicamente attraverso quel mondo".

Avere la possibilità che degli agenti possano essere consci dell'ambiente circostante tramite i valori ottenuti in input dai sensori e poi "agire" utilizzando degli attuatori, ha portato a un elevato sviluppo tecnologico in questo settore, spesso molto vicino all'industria e all'automazione di processi produttivi, riducendo al minimo gli sprechi e la possibilità di errori.
Basta pensare anche a componenti industriali automobilistici che utilizzano bracci meccanici per automatizzare il processo di produzione delle automobili.

Oltre a questo, si possono anche tenere in considerazione utilizzi per lavori ad alta precisione, oppure ancora per il controllo di materiali molto pesanti e il loro spostamento che va attuato con grande attenzione e scrupolo: per questo motivo, tutte le componenti di sensori interagiscono tra loro per arrivare alla riuscita dell'operazione, limitando al minimo gli errori.

2.2.4 Avionica

L'adozione dell'IoT fa parte dell'intero processo di trasformazione digitale delle compagnie aeree: la capacità di sensorizzare e comunicare con sistemi digitali che hanno obiettivi specifici, permette alle compagnie aeree di migliorare in molte aree, sia nel servizio clienti che nella loro gestione interna[11].

Utilizzando tecnologie cloud anche aeroporti minori con costi economici possono essere dotati di una piattaforma informativa che si avvicini a quella dei grandi aeroporti.

Lo sviluppo dell'IoT in questo campo può essere utilizzato per migliorare l'esperienza di volo dei passeggeri come il controllo della temperatura negli ambienti; è possibile migliorare la sicurezza con una diagnostica in tempo reale di tutte le componenti dell'aereo oltre a migliorare l'efficienza di volo come scegliere la rotta più breve per ridurre i costi e massimizzare i profitti.

Inoltre, negli ultimi anni, si è diffuso lo sviluppo di droni a guida autonoma, che fanno uso di tecnologie 4G/5G per il trasferimento dei dati. Vengono utilizzati sistemi avionici molto complessi ma che non si distanziano dalla rete di telecomunicazioni terrestre e il sistema GPS. Allo stesso modo, ci sono velivoli commerciali che sono in grado di decollare e atterrare da soli con sistemi di assistenza alla navigazione tradizionali e con tecniche di intelligenza artificiale.

Considerata la crescente affidabilità delle reti di telecomunicazioni, la loro capillare diffusione sulle aeree continentali e la copertura globale dei sistemi GPS, l'avionica ormai è prossima a una rivoluzione del modo di volare e dell'esperienza di volo del pilota.

2.2.5 Industria automobilistica

Uno dei settori che si stanno evolvendo maggiormente di recente è quello automobilistico.

Particolare sviluppo dell'IoT è avvenuto nel processo produttivo in serie di veicoli: a partire dalla parte robotica come descritto precedentemente, le industrie utilizzano gran parte di sensori e attuatori per monitorare a 360° tutti i processi di produzione, al fine di essere efficienti e di ottimizzare i costi, riducendo gli sprechi e per massimizzare la produttività.

Nei reparti di analisi e sviluppo vengono esaminati i dati per ottenere informazioni per migliorare le auto in produzione. Uno studio efficiente di tutti i parametri può portare a maggiore competitività sul mercato: per questo le grandi aziende stanno investendo molto sull'evoluzione dell'IoT nei processi produttivi e sperimentali.

La gestione della grande mole di dati generata verrà memorizzata sui servizi cloud per poi, grazie a tecniche di machine learning, utilizzarla per migliorare ogni aspetto della costruzione delle automobili stesse.

Altri utilizzi di IoT nel settore automobilistico sono gli aspetti che collegano quelle che vengono definite "Smart Car": questo mercato sta crescendo a ritmi costanti, raggiungendo un valore che ha superato il miliardo di euro. Al suo interno prevalgono i **box GPS/GPRS** per la localizzazione e la registrazione di parametri di guida per finalità assicurative.

In aggiunta, sono stati sviluppati anche sistemi di assistenza alla guida che monitorano l'andamento del veicolo che inviano dei segnali al guidatore in caso di avvicinamento a un altro veicolo o al superamento della carreggiata, o che consentono di ottimizzare il controllo della velocità o abilitano la frenata di emergenza automatica in caso di pericolo; i benefici di questi dispositivi sono

tangibili, in termini sia di risparmio economico che di sicurezza stradale per riduzione del numero di incidenti, vittime e feriti.

Evoluzione di quanto descritto sopra sono le auto a guida autonoma che incorporano l'automazione veicolare rilevando il suo ambiente circostante per muoversi in sicurezza con poco o nessun input umano. Le auto a guida autonoma combinano una varietà di sensori per percepire l'ambiente circostante come telecamere termografiche, radar, lidar, sonar, GPS e unità di misura inerziali: i sistemi di controllo avanzati interpretano le informazioni sensoriali per identificare i percorsi di navigazione appropriati, così come gli ostacoli e la segnaletica pertinente.

Inoltre, anche gli assistenti vocali stanno diventando sempre più comuni all'interno delle automobili: gli smart speaker consentono agli utenti di interagire con il proprio veicolo tramite la voce, ugualmente a quanto accade in ambito domestico.

Ulteriore conferma della diffusione di dispositivi IoT in questo ambito è lo sviluppo della tecnologia *Car2Car* che consente alle automobili connesse di scambiare informazioni con i veicoli vicini oppure con la stessa infrastruttura stradale in merito a manovre repentine e improvvise, e segnalare incidenti o creare schemi di traffico incrociato.

2.2.6 Industria biomedicale

Questo settore comprende tutte le applicazioni in campo medico, come la gestione dei pazienti oppure l'esecuzione di interventi di precisione. Vengono anche utilizzati i dati analizzati per sviluppare nuove tecnologie e nuovi apparati utili per correggere disfunzioni corporali come anche la cura di malattie e lo studio di nuove cure; si parla pertanto di "*Smart Health*" ossia tutti quei sistemi, compresi per esempio gli "*smartwatch*" che tengono costantemente monitorati i parametri vitali e allertano in caso di problemi. Oltre questo si stanno sviluppando anche componenti di sicurezza che monitorano le persone che li indossano per segnalare cadute improvvise e chiamare soccorso tramite connessione a internet o rete cellulare.

2.2.7 Difesa

Per la difesa sono stati coniati i termini **IoMT (*Internet of Military Things*)** o **IoBT (*Internet of Battlefield Things*)**, ossia l'IoT nel contesto militare in cui sono presenti scenari e soluzioni di cyber security. Droni senza pilota e sempre un più armi vengono create con intelligenze artificiali avanzate; oltre questo concetto come difesa si parla anche del difendersi da attacchi informatici per evitare che i sistemi crollino a seguito di questi attacchi. Quindi particolare attenzione si fa sul concetto di sicurezza e sullo scambio di dati controllato per evitare di creare danni diffusi nella rete[12].

È quindi necessario predisporre sistemi di difesa moderni, robusti e implementabili che possano proteggere la rete e le informazioni che circolano in essa.

Attualmente sono allo studio anche ulteriori sistemi fondati, ad esempio, sullo sviluppo dell'AI attraverso algoritmi di apprendimento automatico più robusti basati su reti neurali in grado di riconoscere eventuali tentativi di *deception* o "resistere" ad attacchi informatici ed exploit.

2.2.8 Agricoltura

Anche il settore della "Smart Agriculture" è stato influenzato dall'utilizzo di sensoristica ambientale e territoriale. Per quanto riguarda l'automazione di sistemi di gestione e raccolta, l'IoT può intervenire per controllare vari fattori tra cui migliorare il rapporto tra alimenti prodotti e sostenibilità, argomento che prende il nome di "*agroenergy*".

2.3 Big Data

Sempre più spesso si parla di Big Data ossia di raccolte di dati informativi così estesa in termini di volume, velocità e varietà da richiedere tecnologie e metodi analitici specifici per l'estrazione di valore e conoscenza. Esiste una correlazione quasi naturale fra le tecnologie dell'**Internet of things (IoT)** e dei **big data**: le prime, infatti, sono produttrici di grandi quantità di dati grezzi, mentre le seconde forniscono gli strumenti per filtrare, ordinare e analizzare questi dati ed estrarne valore. La convergenza di IoT e Big Data costituisce quindi un potente strumento che può essere messo a disposizione delle aziende per svariati scopi[13].

Esistono diversi ambiti in cui questi due elementi si combinano, a partire dai singoli processi produttivi fino a diventare la realtà dell'azienda e dei suoi clienti. Applicare questa analisi a un sistema permette di monitorarlo e migliorarlo al fine

di evitare sprechi e per limitare al minimo gli errori. Inoltre, con l'implementazione di attuatori, è possibile creare delle intelligenze artificiali basate su alberi decisionali per adattarsi al mutare delle condizioni esterne.

L'analisi dei dati risulta di grande utilità anche nel settore della **sostenibilità ambientale**. In funzione ***energy management,*** consente l'acquisizione continua e l'analisi dei dati per il monitoraggio dei consumi energetici; in funzione ***energy prediction*** può valutare quanta energia verrà consumata per la produzione, avvalendosi di algoritmi di intelligenza artificiale.

2.4 5G

Per continuare il nostro percorso relativo agli elementi di IoT e alla loro enorme mole di dati che viene generata giornalmente, si rendono necessarie tecnologie all'avanguardia per gestire lo scambio di informazioni con una capacità e velocità molto elevate; per questo è giusto analizzare le caratteristiche della nuova generazione 5G, considerata un ottimo supporto per la crescita del mondo IoT.

Nella seguente figura sono evidenziate le principali migliorie che la nuova generazione di trasmissione dati porterà:

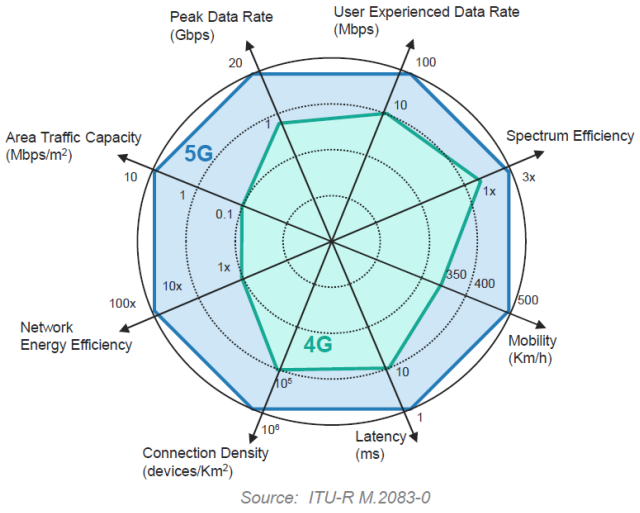

Figura 2: differenza tra caratteristiche 4G e 5G

Come si vede rispetto alla quarta generazione, gli obiettivi del 5G sono di migliorare la tecnologia esistente in ogni ambito. La principale differenza di questa tecnologia è superare la mentalità del "*one-fits-all*" delle tecnologie precedenti: si vuole creare una rete adattabile alle situazioni in grado di dare le massime performance nel luogo e con gli obiettivi con cui è costruita. Per questo vengono gestiti molti processi anche con tecniche di machine learning e intelligenza artificiale in grado di massimizzare tutti i parametri in tempo reale per ottenere la migliore connessione possibile.

2.5 Machine Learning

Il **Machine Learning (ML)** è una tecnica di apprendimento usata per "insegnare" ai computer e ai robot a compire azioni ed attività in modo naturale come gli esseri umani o gli animali. A partire dall'analisi dei dati a disposizione, si generano sistemi che apprendono imparando dall'esperienza in modo automatico senza partire da equazioni predeterminate.

Gli algoritmi di ML usano metodi matematico-computazionali per acquisire informazioni direttamente dai dati, migliorando le loro prestazioni in modo "adattivo" mano a mano che gli "esempi" da cui apprendere aumentano. Tutti questi algoritmi partono dai dataset delle misurazioni ottenute tramite sensori o altri tipi di dati che vengono raccolte e fornite agli algoritmi per la fase di training.

Il seguente progetto di tesi è impostato per caricare e fornire i dataset e le misurazioni presenti sul database MongoDB dell'Università utili per generare modelli e predizioni finalizzati allo sviluppo di algoritmi di machine learning.

Analizziamo in breve i principali tipi di machine learning:

2.5.1 Supervised learning

Questa categoria di Machine Learning è definita a partire dall'uso di set di dati etichettati per addestrare algoritmi in grado di classificare i dati o prevedere i risultati in modo accurato. I dataset contengono sia le informazioni dei valori di input sia quelle relative ai risultati desiderati (output). L'obiettivo è che il sistema identifichi una regola generale che colleghi i dati in ingresso con quelli in uscita in modo da poter poi riutilizzare tale regola appresa in ambiti simili.

Come scrive **Adam Geitgey, sviluppatore software, nel suo articolo "Machine Learning is Fun!"** a proposito del Supervised Learning:

«***Nell'apprendimento supervisionato il lavoro di risoluzione viene lasciato al computer. Una volta compresa la funzione matematica che ha portato a risolvere uno specifico insieme di problemi, sarà possibile riutilizzare la funzione per rispondere a qualsiasi altro problema similare***»,

2.5.2 Unsupervised learning

In questa seconda categoria di Machine Learning, al sistema vengono forniti solo set di dati senza alcuna indicazione del risultato desiderato (senza output). Lo scopo di questo metodo di apprendimento è "risalire" a schemi e modelli nascosti, ossia identificare negli input una struttura logica senza che questi siano preventivamente etichettati.

Oltre a quelli elencati ci sono anche altri tipi di machine learning a partire dai dati misurati, come quello *reinforced*, semi-supervisionato o probabilistico.

2.6 Framework IoT

La sempre più estesa raccolta di dati dai dispositivi IoT sottolinea la necessità di strumenti di sviluppo e di applicazioni efficienti. I servizi cloud IoT allo stato dell'arte sono potenti, ma le migliori soluzioni sono proprietarie, e questo si pone in contrasto con la crescente domanda di interoperabilità e standardizzazione: Measurify nasce come framework open source, che supporta in particolare la ricerca collaborativa, sfruttando lo stato dell'arte delle tecnologie di gestione dei dati, e puntando a uno sviluppo efficace ed efficiente; numerosi sono i benefici di non essere legati a una piattaforma commerciale e in aggiunta, i tempi di configurazione necessari per iniziare un progetto da zero sono di gran lunga ridotti. Concentrandosi sul concetto di misurazione, è stata generata un'architettura applicabile a una varietà di domini e contesti[14].

La letteratura offre un'ampia copertura di framework IoT. Cheruvu et al. [15] forniscono un sondaggio che categorizza tali strumenti in base a un focus sul

consumatore, sull'industria o sulla gestibilità. Atamani et al. [16] forniscono una revisione di diversi framework e piattaforme IoT disponibili, analizzando criteri come la sicurezza, l'analisi dei dati e il supporto della visualizzazione.

Soluzioni potenti sono attualmente disponibili sul mercato per diversi tipi di servizi IoT: il servizio cloud Amazon Web Services for the IoTs (AWS IoTs) [17] presenta moduli di architettura per la gestione dei dati, la connettività e il controllo dei dispositivi, l'analisi e il rilevamento degli eventi. Uno stack di funzioni è disponibile sul back-end: DynamoDB, Kinesis, Lambda, S3, SNS, SQS, ecc. Poiché AWS è un fornitore di servizi cloud, numerosi servizi di elaborazione dati sono già integrati. L'integrazione delle applicazioni nella piattaforma è supportata da vari servizi di messaggistica e accodamento; tuttavia, ci sono limitati attributi personalizzati per le cose, il che sfida la gestibilità e aumenta la latenza [18]: AWS IoT non distingue tra sensori, attuatori e dispositivi, poiché si concentra sul concetto di *cose*.

Microsoft fornisce come servizio la piattaforma Azure [19] che permette agli sviluppatori di creare programmi basati sul cloud utilizzando una piattaforma commerciale di software. Azure Sphere contiene strumenti per i kit di sviluppo software (SDK) edge [20] per consentire una connettività sicura edge-to-cloud.

Altri framework commerciali [21], come Google Cloud e Bluemix, hanno una varietà di servizi cloud IoT; nonostante le loro differenze in termini di prestazioni ed efficienza nella pratica, Zúñiga-Prieto et al. [22] indicano i lunghi tempi di implementazione come un problema condiviso tra questi framework.

Diversi framework di dati IoT sono stati presentati in letteratura per affrontare specifici domini applicativi: esempi recenti riguardano la medicina legale [23], le case intelligenti [24] e le città intelligenti [25]. In un approccio più generale, Jiang et al. [26] hanno proposto un framework che affronta le sfide tipiche dell'IoT (grande volume di dati, diversi tipi di dati, dati a generazione rapida, requisiti complicati, ecc.): per i dati strutturati, propongono un modello di gestione del database che

combina ed estende più database e fornisce API di accesso unificate; per i dati non strutturati, il framework avvolge ed estende il file system distribuito di Hadoop basato sul modello di file repository per implementare la gestione delle versioni e l'isolamento dei dati multitenant. Un modulo di configurazione delle risorse supporta la gestione statica e dinamica dei dati in termini di meta-modello predefinito, cosicché le risorse di dati e i servizi correlati possono essere configurati in base ai requisiti del tenant.

Più recentemente, Cai et al. [27] hanno presentato un quadro funzionale che identifica le aree di acquisizione, gestione, elaborazione ed estrazione dei big data dell'IoT in cui diversi moduli tecnici associati sono definiti e descritti in termini di caratteristiche e capacità chiave. Fu et al. [28] si sono occupati in particolare dell'efficienza e della sicurezza della memorizzazione dei dati IoT, proponendo un quadro che mantiene i dati sensibili al tempo (ad esempio, le informazioni di controllo) sul *edge* e invia i rimanenti (ad esempio, i dati di monitoraggio) al cloud.

Analogamente ad Atmosphere, Paganelli et al. [29] propongono un framework che supporta gli sviluppatori nel modellare le *cose* intelligenti come risorse web: il framework supporta la definizione e la progettazione del tipo di risorsa, un software generico per le operazioni sulle risorse web, una mappatura tra risorse web e fonti di dati, e strumenti di programmazione e pubblicazione. La valutazione dell'utente ha riguardato l'uso di IDN-studio (InterDataNet-studio), applicazione web con interfaccia grafica per progettare e gestire risorse web e per personalizzare e migliorare la rappresentazione di un punto di interesse all'interno dell'applicazione mySmartCity, valutando il suo rendering eseguito tramite IDN-viewer.

Sharma e Wang [30], individuano quattro caratteristiche principali dei dati IoT nelle piattaforme cloud: alta eterogeneità *multisource*, enorme scala dinamica, basso livello con semantica debole, imprecisione; queste caratteristiche sono importanti, in quanto evidenziano le peculiarità che dovrebbero essere fornite da un efficace

framework di dati IoT (ad esempio, la caratterizzazione delle fonti, la varietà di configurazioni/aggregazione dei dati di origine, il calcolo degli outlier [31]), che è stato tenuto in considerazione nella progettazione di Atmosphere.

3. Strumenti di sviluppo

3.1 Measurify

Measurify è un'API per la gestione delle *smart things* negli ecosistemi IoT basata sul cloud, astratta e orientata alla misurazione; precedentemente chiamata Atmosphere IoT Framework [14], si concentra sul concetto di misurazione, ambito molto comune nell'IoT, che rende più facile ed efficace il processo di astrazione necessario per indirizzare diversi domini e contesti operativi.

Measurify è un progetto ideato, sviluppato e mantenuto da Elios Lab dell'Università di Genova. Wondertech contribuisce allo sviluppo e alla manutenzione del codice sorgente. Al fine di supportare la comunità di sviluppatori IoT, è rilasciato open source sotto licenza MIT.

La seguente API è stata progettata per rappresentare il contesto applicativo e i suoi elementi come oggetti software interconnessi, su cui costruire applicazioni. Questi oggetti sono modellati come risorse, con modelli e funzionalità proprie, accessibili attraverso una serie di interfacce RESTful.

Al centro di queste risorse ci sono gli elementi essenziali che sono comuni nell'ambiente IoT: Thing, Feature, Device e Measurement: una Thing rappresenta l'oggetto di una misurazione; una Feature descrive la quantità (tipicamente fisica) misurata da un Device; ogni elemento di una Feature ha un nome e un'unità; un Device è uno strumento che fornisce misurazioni riguardanti una Thing; infine, una

Measurement rappresenta un valore di una Feature misurato da un Device per una specifica Thing.

La figura seguente mostra le relazioni tra le risorse, evidenziando il ruolo centrale del concetto di misurazione: nello specifico, è raffigurato un semplice esempio nel contesto della raccolta di informazioni meteorologiche (es. temperatura e velocità del vento).

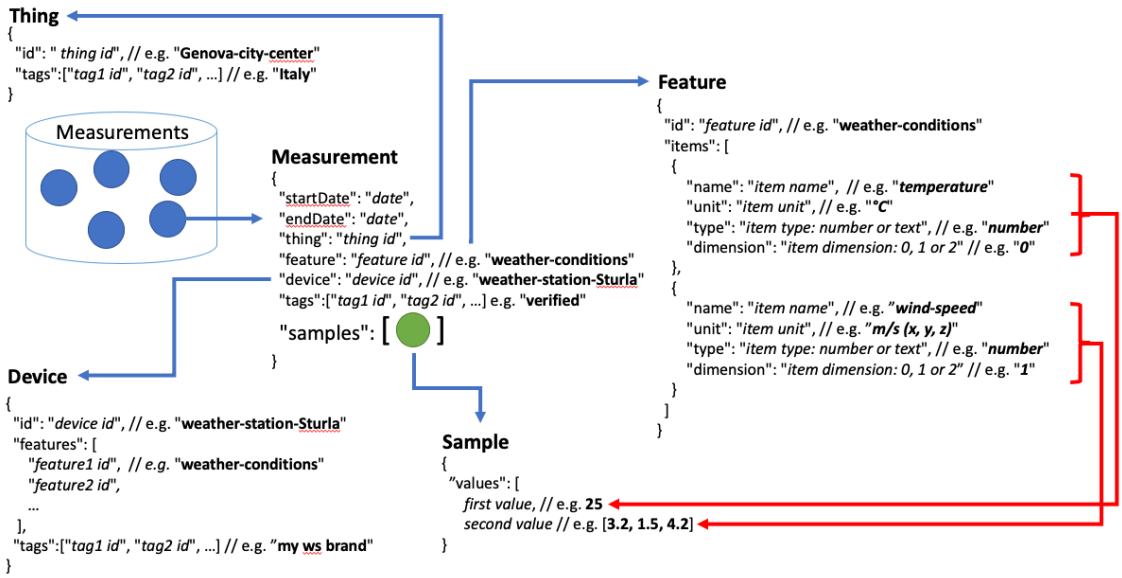

Figura 3: Caratteristiche misurazioni di Measurify

Il concetto di Misura astrae i valori inviati e recuperati dal Cloud. La sua struttura deve corrispondere al tipo di caratteristica misurata. Ogni misurazione è un vettore di campioni: possono essere campioni raccolti in tempi diversi (presi a intervalli specificati dal campo "delta"), un singolo valore o un insieme di informazioni statistiche (ad esempio, media, stdev, ecc.). Ogni campione può essere uno scalare (per esempio una temperatura), un vettore (per esempio l'orientamento nello spazio) o un tensore di numeri (per esempio, punti di dati multidimensionali generali).

Nell'esempio precedente, abbiamo solo un singolo campione scalare per la misurazione.

La risorsa Feature viene utilizzata per convalidare l'array di valori di ogni misurazione ricevuta: questo è mostrato nella figura precedente, dove gli attributi "type" e "dimension" (0 per gli scalari, 1 per i vettori, 2 per i tensori) dalla risorsa Feature devono corrispondere rispettivamente al tipo di array di valori del Sample e alla sua profondità.

Altre risorse supplementari sono User, Log, Login, Script, Tag, Constraint e Computation.

3.2 Notebook

Il Jupyter Notebook o Colab Notebook sono applicazioni web open source che possono essere usati per creare e condividere documenti testuali interattivi che contengono codice eseguibile, equazioni, grafici e testo. Questa possibilità rende i documenti molto più *human-readable* e di facile comprensione per chi successivamente dovrà interpretare i passaggi eseguiti.

Jupyter è diventato il più utilizzato dai data scientist poiché offre la possibilità di fare *data cleaning & trasformation*, simulazioni numeriche, modellazione statistica, machine learning e molto altro. L'utilizzo di questi notebooks è molto utile nel caso di processi lavorativo in team, fondamentale in molti settori. La soluzione basata sull'uso dei Notebooks crea un continuo collegamento tra codice di programma e testo esplicativo, consentendo all'utente di scambiare e creare in tempo reale codici, equazioni, visualizzazioni, includendo le informazioni esplicative.

Inoltre tramite Apache Spark[32] è possibile eseguire applicazioni su piattaforme Big Data.

3.3 Python

Python è un linguaggio di programmazione interpretato, orientato agli oggetti e di alto livello con semantica dinamica. Le sue strutture di dati di alto livello, combinate con la tipizzazione dinamica e il *binding* dinamico, lo rendono molto attraente per lo sviluppo rapido di applicazioni. La sintassi semplice e facile da imparare di Python enfatizza la leggibilità, riducendone il costo della manutenzione del programma. Python supporta moduli e pacchetti, il che incoraggia la modularità del programma e il riutilizzo del codice. L'interprete Python e la vasta libreria standard sono disponibili in forma sorgente o binaria senza costi per tutte le principali piattaforme, e possono essere distribuiti liberamente; le librerie di Python non sono altro che **collezioni di metodi e funzioni** che permettono di svolgere azioni senza scrivere il codice di ogni passaggio. Questo si rivela molto utile poiché permette di semplificare enormemente il codice, alleggerendo l'implementazione a mano di numerose operazioni.

4. Implementazione

In questa sezione verranno illustrate le principali funzioni che sono state sviluppate per la gestione e lo scambio dati attraverso la API Measurify. In primo luogo, verranno discusse le particolarità della libreria Python *measurify.py*, necessaria per i collegamenti tra la API e il notebook Python, per poi passare alla implementazione di nuove funzionalità su Measurify al fine di migliorare le prestazioni e permettere la gestione di una quantità elevata di dati.

4.1 Libreria measurify.py

Come anticipato in precedenza, la libreria Python dovrà essere installata all'interno dell'ambiente notebook o editor di codice sorgente per poter essere utilizzata. Il file si trova nella sezione *Dataset* [33] all'interno del Github di Measurify [34].

Questa libreria è stata sviluppata con l'obiettivo di poter accedere alla API Measurify tramite codice in Python, semplificando e ottimizzando i passaggi della comunicazione. Utilizzando questa libreria all'utente non sarà più richiesta una conoscenza approfondita della API e di tutte le sue rotte ma può caricare e ottenere dati sfruttando le funzioni di quest'ultima.

Il workflow dei passaggi principali di questa libreria può essere riassunto nella figura seguente:

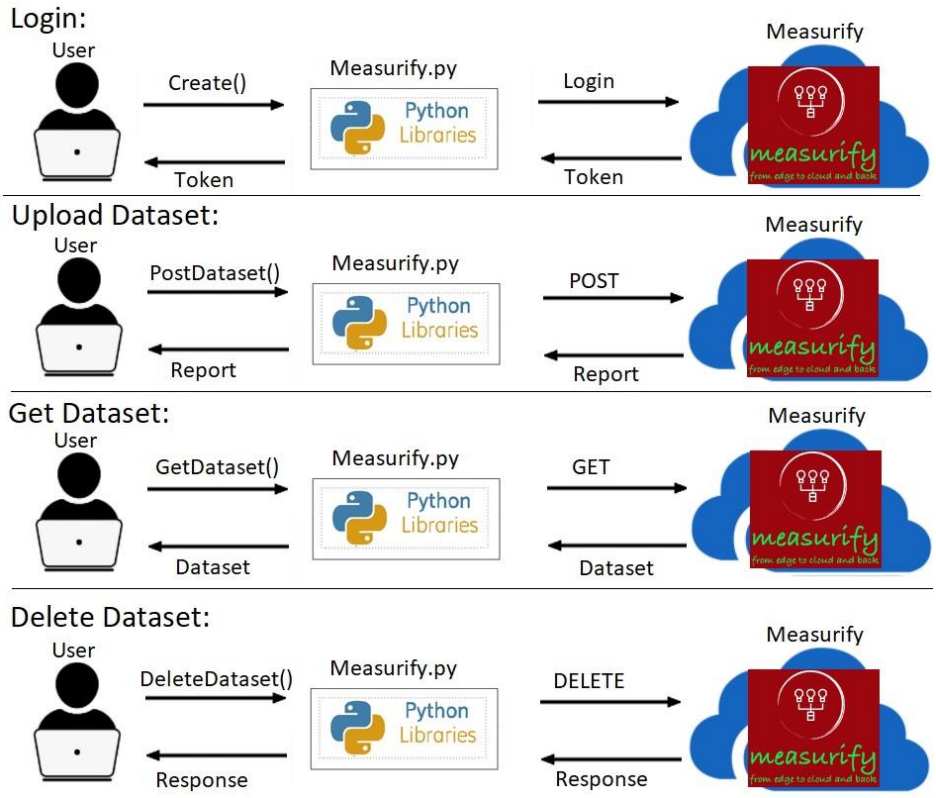

Figura 4: workflow della libreria measurify.py

Nell'immagine, il primo passaggio è quello di login in cui viene utilizzata la funzione Create della libreria che effettuerà tutti i passaggi di autenticazione alla API Measurify; se il login è andato a buon fine, si riceverà un token che verrà utilizzato dalla libreria per effettuare le chiamate successive.

Il secondo punto imposta i passaggi per caricare un proprio dataset sul database utilizzando la PostDataset che verrà gestita come una chiamata POST al server: il body della Post conterrà il file delle misurazioni e il file di *description*: in risposta ad esso otterremo un Report contenente le informazioni di quali misure sono state salvate correttamente e quali no.

Per ottenere il Dataset sarà necessario utilizzare la funzione GetDataset impostando i filtri e il formato: la libreria eseguirà una GET a Measurify con tutte le info necessarie e in risposta ad esso riceveremmo il Dataset.

L'ultimo passo è quello per cancellare un Dataset utilizzando la funzione DeleteDataset che userà una rotta di tipo Delete e otterrà in risposta l'avvenuta cancellazione.

Nelle figure seguenti si vedrà nel dettaglio il formato della richiesta e della risposta di queste funzioni.

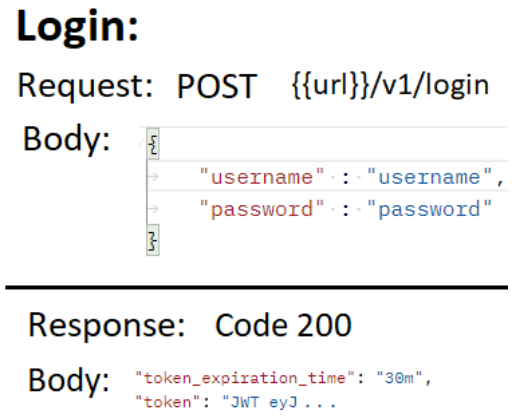

Figura 5: Richiesta e risposta della Login

Nella Figura 5 la richiesta della Login corrisponde ad una Post alla *route* di login di Measurify, aggiungendo come *body* l'username e la password. Se la richiesta ha avuto esito positivo verrà inviata come risposta il codice 200 e il valore del token utile per le chiamate successive.

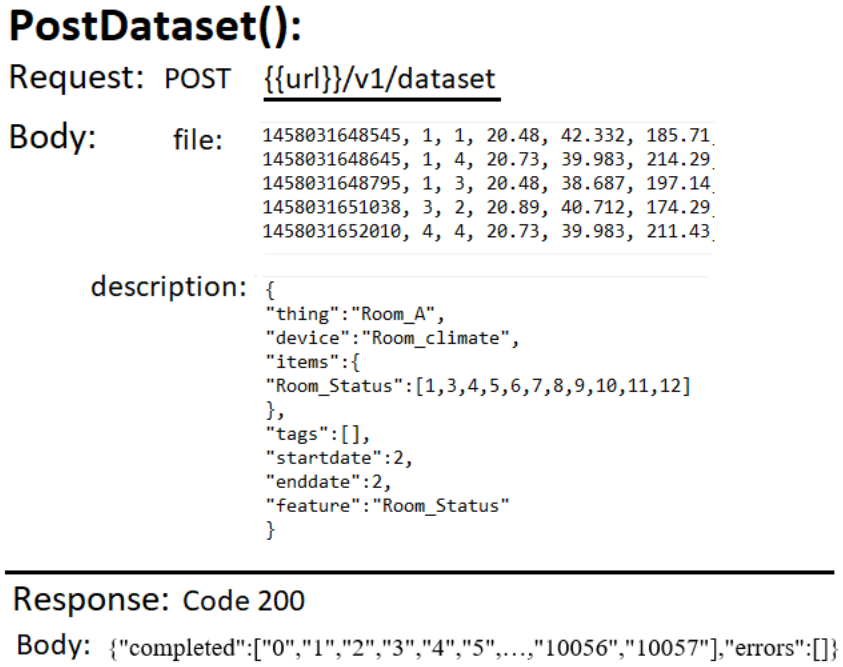

Figura 6: Richiesta e risposta della PostDataset.

Per quanto riguarda la PostDataset la *route* a cui bisogna inviare i dati è */v1/dataset* e il body deve contenere il file contenente le misurazioni e il file di *description*. Come risposta arriverà il report con tutte le misurazioni che sono state salvate correttamente e quali invece contengono degli errori; nel caso tutte le misurazioni siano state registrate correttamente il codice sarà 200, altrimenti sarà 202.

GetDataset():

Request: GET {{url}}/v1/dataset

Parameter: Format: CSV, CSV+, DATAFRAME

Force: True/False

Filter

Response: Code 200

Body: CSV

device	samples
CENTRALINA	[35, 987792]
CENTRALINA	[23, 994543]
CENTRALINA	[23, 1001893]
CENTRALINA	[12, 1001895]
CENTRALINA	[5, 1008943]
...	...
CENTRALINA	[14, 1144531]
CENTRALINA	[28, 1148531]
CENTRALINA	[31, 1152107]
CENTRALINA	[30, 1152108]
CENTRALINA	[28, 1152111]

CSV+

```
"device","EID","RelT"
"Room_climate","8184","8177"
"Room_climate","8183","8176"
"Room_climate","8182","8176"
"Room_climate","8181","8174"
"Room_climate","8180","8173"
"Room_climate","8179","8172"
"Room_climate","8178","8172"
"Room_climate","8177","8171"
"Room_climate","8176","8169"
"Room_climate","8175","8169"
```

DATAFRAME

	_id	visibility	tags	...
0	PM10	private	[centraline-qualita-aria]	...
1	PM10	private	[centraline-qualita-aria]	...
2	PM10	private	[centraline-qualita-aria]	...
3	PM10	private	[centraline-qualita-aria]	...
4	PM10	private	[centraline-qualita-aria]	...
..	...	...	...	...
136	PM10	private	[centraline-qualita-aria]	...
137	PM10	private	[centraline-qualita-aria]	...
138	PM10	private	[centraline-qualita-aria]	...
139	PM10	private	[centraline-qualita-aria]	...
140	PM10	private	[centraline-qualita-aria]	...

Figura 7: Richiesta e risposta della GetDataset.

La GetDataset viene utilizzata per richiedere alla API un particolare Dataset. Questa funzione imposta una Get alla *route dataset*, mettendo come parametri il formato richiesto, se attivare Force e i parametri del filtraggio. Il codice di risposta sarà 200 e conterrà i dati filtrati nel formato richiesto.

DeleteDataset():

Request: DELETE {{url}}/v1/dataset/{{filename}}

Response: Code 200

Body:


```
"size": 1085622,
"results": "{\"completed\":[\"0\",\"1\",\"2\"...
          \"10056\",\"10057\"],\"errors\":[]}",
"_id": "centraline-qualita-aria",
"owner": "61b29d6d6a565b309cbb45ed",
"timestamp": "2022-03-20T19:34:00.058Z",
"lastmod": "2022-03-20T19:34:40.635Z"
```


Figura 8: Richiesta e risposta della DeleteDataset.

Per cancellare un Dataset che è stato caricato precedentemente è necessario eseguire una Delete alla rotta *dataset* aggiungendo alla fine il nome del file da cancellare. Se è stato cancellato correttamente, la risposta conterrà il codice 200 e il body conterrà le informazioni di quanto è stato cancellato.

Riassumendo le principali funzioni di questa libreria sono Create che serve per la configurazione della libreria e impostare i valori di login e le tre funzioni che verranno analizzata a fondo PostDataset, GetDataset e DeleteDataset.

Le sfide affrontate in questa tesi sono state quelle di creare una modalità di caricamento dati che sia slegata dal settore in cui sono state prese le misurazioni in modo da avere la maggior flessibilità possibile per adattarsi a qualsiasi tipo di dataset; questo è stato portato a termine tramite la possibilità data all'utente di creare un file *description* per far comprendere alla API il modo in cui sono organizzati i dati all'interno del CSV.

4.2 Installazione

Per poter installare la libreria all'interno del notebook è necessario caricare i file della libreria nell'ambiente di sviluppo.
Il file principale è *measurify.py* contenente tutte le funzioni principali della libreria, mentre il file *Measurify_Dictionary.py* è necessario alla libreria in quanto contiene tutte le informazioni testuali utilizzate durante i collegamenti in caso di eccezioni o errori. Infine, *measurifyConfig.json* è il file di configurazione dei parametri e delle rotte alla API: questo file può essere modificato dall'utente quando si vogliono creare in parallelo più istanze della libreria, modificando l'URL di accesso alla API oppure collegandosi con un username e password differenti.

```
{
    "url":"https://localhost/",
    "username" : "admin",
    "password" : "admin",
    "login_route":"v1/login",
    "things_route":"v1/things",
    "features_route":"v1/features",
    "devices_route":"v1/devices",
    "measurements_route":"v1/measurements",
    "dataset_route":"v1/dataset",
    "dataset_info":"/info",
    "file":"C:\\Users\\matte\\Postman\\files\\csvtest.txt",
    "description":"C:\\Users\\matte\\Postman\\files\\Allegato.txt"
}
```

È fortemente sconsigliato cambiare il percorso delle rotte al server in quanto potrebbe non funzionare più la libreria, mentre gli altri campi possono essere modificati per ampliare le funzionalità di Measurify.

4.3 Utilizzo

4.3.1 Funzione Create

Per utilizzare la libreria bisogna importarla con il suo nome; per effettuare la prima connessione alla API, è necessario utilizzare la funzione *Create*.

```
import measurify

api=measurify.Create()
```

Questa funzione oltre all'inizializzazione della libreria, effettua il login a Measurify ottenendo il token necessario per le comunicazione successive.

La funzione *Create* ha diversi parametri opzionali:

- config_Directory con valore di default “measurifyConfig.json” contiene la *path* del file di configurazione.
- *username* e *password* sono campi opzionali che possono essere inseriti come parametro nella funzione Create oppure verranno letti dal file di configurazione: è consigliato l’uso del file di configurazione.
- *options* viene utilizzato per modificare alcuni parametri del file di configurazione passando come parametro una stringa contenente le chiavi e i nuovi valori che si vogliono utilizzare. Questo si dimostra utile per creare diverse istanze della libreria con differenti parametri.
- verbose è il parametro utilizzato per ottenere più informazioni dalla libreria come i dettagli della comunicazione con il server.

Esempi di utilizzo:

- api=measurify.Create(): senza parametri utilizzando le informazioni di default e il file di configurazione.
- api=measurify.Create(config_Directory="path"): Per indicare una path diversa da quella di default al file di configurazione.
- api=measurify.Create(username="username",password="password"): password e username vengono inseriti manualmente.
- opt1='{"username":"username1","password"="password1","url"="url1"}'
  opt2='{"username":"username2","password"="password2","url"="url2"}'
  api1=measurify.Create(options=opt1)
  api2=measurify.Create(options=opt2)
  Utilizzato per avere due istanze della libreria con parametri differenti.
- api=measurify.Create(verbose=True): per attivare le informazioni verbose.

4.3.2 Filter

In questa sezione vediamo come impostare i filtri alle chiamate di Measurify.

Per dare maggiore flessibilità all'utente, abbiamo scelto due modalità per impostare i filtri: la prima è quella di inserire il filtro direttamente, scrivendo la stringa corrispondente al filtro che si vuole utilizzare; in alternativa, è possibile usare la funzione createFilter().

Nel primo caso inserisco direttamente la stringa del mio filtro che voglio utilizzare: questa possibilità è la più flessibile in quanto un utente può recuperare tutte le funzionalità del filtraggio nella documentazione di Measurify e utilizzarla.

Esempio: filter={"thing"="thing1","feature"="feature1","device"="device1"}

La seconda soluzione è semplificata per chi non conosce a fondo l'architettura della API: i parametri della funzione sono *thing, feature e device* e per una esecuzione corretta almeno uno di questi tre parametri deve essere inserito.

Esempi:

- filter=api.createFilter(thing=" thing1"): impostata solo la thing.
- filter=api.createFilter(feature=" feature1"): impostata solo la feature.
- filter=api.createFilter(device=" device1"): impostato solo il device.
- filter=api.createFilter(thing="thing1",feature="feature1"): impostate thing e feature.
- filter=api.createFilter(thing="thing1",feature="feature1",device="device1"): impostati tutti e tre i parametri.

4.3.3 Set

Esistono due funzioni per impostare da codice le variabili Verbose e Filter per utilizzarle nelle funzioni successive senza passarle come parametro.

- api.setVerbose(verbose=True): per impostare che da lì in poi voglio ottenere le comunicazioni verbose, oppure False nel caso voglia disattivarle.
- api.setFilter(filter): per impostare il filtro da utilizzare nelle chiamate successive.

4.3.4 GetDataset

Questa è la funzione più importante della libreria ed è stata creata per dare la maggior flessibilità possibile all'utente che la utilizza: tramite essa, possiamo ottenere **tutte** le misurazioni presenti sul database che rispecchiano i requisiti del filtro dell'utente. La parte delle *route* alla API è gestita dalla libreria e ciò permette ad un utente che non conosce Measurify di interfacciarsi con esso senza difficoltà.

I parametri di getDataset sono format, filter, verbose, filename, downloadPath, limit, page.

Format

Tramite il parametro format è possibile scegliere in che formato ottenere le misure:

- CSV: di default, le misure vengono inviate in formato CSV in quanto è il più utilizzato e ottimizzato per lo scambio di questo genere di dati, presentando i dati in forma di tabella.
- CSV+: Questo formato è stato progettato e implementato per l'architettura delle misurazioni, scelta dalla API Measurify, la quale è basata sulla suddivisione delle misurazioni in feature, thing e device. In questo formato

se la feature è fissata, la colonna dei valori verrà divisa in ulteriori colonne in base a quanti argomenti ha la feature (esempio: in una misurazione nello spazio 3D il valore salvato sul database sarebbe del tipo [x,y,z] e nel csv la colonna "*values*" conterrebbe [10,20,30]; nel caso del CSV+, verranno generate le colonne "x", "y" e "z" al posto della colonna "*values*" disponendo all'utente già i valori suddivisi). N.B: Questa funzionalità avviene solo nel caso in cui la feature è fissata all'interno del filtro della richiesta altrimenti si otterrà un CSV normale.

- PANDAS DATAFRAME: è stato scelto di utilizzare il formato *Pandas dataframe* perché il suo impiego è molto diffuso dai data scientist per le analisi dei dati. Questo permette di ottenere tutti i dati già pronti per la libreria Python denominata Pandas e di disporne liberamente.
- JSON: Questo è il formato utilizzato originariamente dalla API per lo scambio di misurazioni ma non è ottimale per la gestione di un elevato numero di dati.

Filter

Come detto precedentemente è riferito al filtro che andrà inserito nella richiesta

Verbose

Abilita o disabilita la funzione verbosa già descritta.

Filename

Questo parametro è molto importante perché è stata data la possibilità alla API di caricare sul database i dataset presenti all'interno di file CSV: nel momento dell'inserimento del dataset sul database, ogni misurazione avrà aggiunto il tag che lo assocerà al nome del dataset in cui era presente, prendendo il nome del file CSV

da cui è stato estrapolato. Inserendo il parametro filename, si vorrà ottenere tutte le misurazioni che possiedono quel tag ossia che sono state caricate dallo stesso file CSV; questo senza ulteriori filter permette di riottenere il CSV originario e in caso di ulteriore filtraggio permette di ottenere solo la parte voluta di quel file.

DownloadPath

Questo parametro serve per indicare il percorso con aggiunto il nome del file da generare, in cui si vuole inserire tutte le misurazioni ottenute. Questa variabile è opzionale e nel caso non sia inserita non avviene alcun scaricamento dati; nel caso in cui sia inserita una *path* valida, al momento della ricezione dati, verrà generato un file indicato nel parametro e verranno inseriti tutti i dati nel formato scelto. Questo è utile per salvare in locale il dataset e non doverlo richiedere ogni volta.

Limit e page

Questi due parametri vengono usati da Measurify per cambiare il numero di elementi a cui si vuole accedere e possono essere usati dall'utente per ottenere il dataset fino a un certo numero di elementi fissato.

4.3.5 GetDatasetInfo

Questa funzione è stata creata con lo scopo di ottenere le informazioni su un dataset presente all'interno del database; è sufficiente inserire il nome del dataset da rilevare e sarà possibile ottenere informazioni come la dimensione, la data di caricamento e il report di quando è stato caricato per verificare la presenza di errori in alcune righe o se alcune di esse non sono state salvate sul database.

4.3.6 PostDataset

Questa funzione è utilizzata per la POST delle misurazioni sul database a partire da file con il dataset in formato CSV o TXT. Il file sarà inviato e ogni misura presente verrà organizzata e salvata all'interno del database; successivamente, verrà restituito un report che terrà conto di tutte le righe che sono state inserite correttamente e quali hanno riscontrato errori dovuta a parametri errati o altro.

Il parametro "file" di questa funzione necessita della *path* del documento che, seguendo le specifiche del formato CSV conterrà in ogni riga una misura differente in cui ogni parametro è separato da una virgola.

È stato deciso di allegare un secondo file contente tutte le informazioni di come sono classificate le informazioni all'interno del file CSV: l'idea è che alle API si possa passare un file CSV e che il server lo inserisca all'interno del database sotto forma di things, features, devices e measurements. Per poter far questo e avere maggiore flessibilità, oltre al file CSV, è necessario fornire alle API un file JSON di configurazione che spieghi alle API il significato delle diverse colonne del CSV: per allegare questo file basta inserire la *path* nel parametro "*description*" di questa funzione.

Si può vedere ogni singolo componente a quale colonna appartiene.

```
{
"thing":2,
"device":4,
"items":{
   "test-feature":[6]
},
```

"tags":[3],

"startdate":1,

"enddate":5,

"feature":"feature-test"

}

Nel caso in cui alla chiave venga corrisposto un numero, la API riconoscerà il campo della colonna corrispondente con la chiave indicata, mentre se è presente una parola, allora quella chiave sarà fissata per tutte le misurazioni presenti nel dataset. Questa possibilità aiuta nel caso non tutti i campi da salvare nel database siano presenti all'interno del dataset che è preso in considerazione. Nonostante all'inizio del percorso della tesi si era scelto un'ottica in cui un dataset poteva contenere una sola feature di dati (per esempio un dataset poteva contenere solo misurazioni della temperatura oppure solo le coordinate nello spazio), la versione definitiva del codice ha deciso di dare la possibilità di inserire features diverse inserendo nel dataset il campo "feature". Per questo motivo il campo "items" che corrisponde ai valori della misurazione associa il nome delle feature alle colonne corrispondenti.

Nel paragrafo dei risultati sperimentali si vedrà questa funzione con degli esempi di dataset.

Force

Un parametro molto importante di postDataset è Force che di default è disattivato. La API permette il salvataggio di una misura sul database nel caso in cui gli oggetti Thing, Device e Feature si ritrovino già all'interno del database; nel caso in cui per esempio la Thing non sia registrata nel database, non verrebbe accettata la misurazione e verrebbe rifiutata dal server con un messaggio di errore.

Questo porta l'utente a dover registrare questi tre parametri prima di passare al caricamento delle misure. La Feature è l'elemento centrale della misura perché indica il tipo di parametri che vengono salvati, numeri o stringhe, e la quantità di elementi che possiede la feature corrente, dando un nome ad ogni valore. Questo valore dovrà sempre essere inserito a priori perché è necessario per controllare che la misurazione sia avvenuta correttamente.

Dall'altra parte, Thing e Device contengono informazioni su come sono state ottenute queste misurazioni. In un dataset che contiene numerosi dati, il passaggio di inserire manualmente ogni Thing e Device potrebbe risultare molto laborioso e dispendioso in termini temporali; pertanto, è stato deciso di permettere la creazione automatica con valori di default di questi oggetti. Per attivarla basta impostare il parametro force a *True* cosicché ogni volta che compariranno nuove Thing o Device, essi verranno creati sul database così da consentirne la misurazione.

Verbose

Anche qui è possibile abilitare la funzione verbose per ottenere più informazioni riguardanti le comunicazioni al server.

Report

Come output della funzione postDataset, un "report" del salvataggio delle misurazioni è disponibile per l'utente: esso conterrà come prima parte le misurazioni "*completed*" ossia il numero della riga delle misurazioni salvate con successo, mentre nella seconda parte si avrà "*errors*" ossia quali righe hanno riscontrato errori con anche una breve linea di testo che lo descriva. Questo è utile perché si possono andare a correggere le righe non corrette al fine di avere un database sempre corretto.

4.3.7 DeleteDataset

Duale al postDataset, questa funzione elimina dal database un dataset caricato precedentemente con tutte le sue misurazioni. Unico parametro da inserire è quello del nome del file da eliminare e con esso verrà cancellato tutti gli elementi correlati al file.

4.4 Measurify Javascript

Questa sezione presenta le nuove feature della API Measurify che sono state introdotte parallelamente allo sviluppo della libreria Python per migliorare le prestazioni del framework.

Per prima cosa sono stati studiati diversi modi per migliorare l'efficienza nell'invio dei dati e si è deciso di implementare nuovi formati: l'obiettivo era quello di modificare la risposta del server alla Get dei dati per ottimizzare tempi e risorse sia dal lato server che dal lato client.

Nella risposta originale del server, i dati venivano inviati elemento per elemento in formato json:

{"docs": [

{

"visibility":"private",

"tags":[],

"_id":"618c1210147bce00dc799957",

"startDate":"2021-11-10T18:40:16.232Z",

"endDate":"2021-11-10T18:40:16.232Z",

"thing":"car-test",

"feature":"position",

"device":"GPS",

"samples":[{

"values":[170,90,30]}]

},

{

"visibility":"private",

“tags”:[],

"_id":"618c120d147bce00dc799955",

"startDate":"2021-11-10T18:40:12.925Z",

“endDate”:”2021-11-10T18:40:12.925Z”,

"thing":"car-test",

“feature”:”position”,

"device":"GPS",

"samples":[{

"values":[170,70,20]}]

}…

L’invio in questo formato di 17 misurazioni comportava l’uso di 2468 caratteri corrispondente a 2468 bytes.

Si è deciso di organizzare i dati in un formato diverso raggruppando tutti i campi uguali di ogni singola misurazione senza ripetere il nome del campo per ogni misura.

Questa è la risposta modificata del server in cui avviene l'invio blocchi di dati:

{"docs":

[{"visibility":["private","private","private","private","private","private","private","private","private","private"],

"tags":[[],[],[],[],[],[],[],[],[],[]],

"_id":["618c1210147bce00dc799957","618c120d147bce00dc799955","618c1201147bce00dc799953","618c11fd147bce00dc799951","618c11f0147bce00dc79994f","618c11e9147bce00dc79994d","618c11e6147bce00dc79994b","618c11d2147bce00dc799949","618c11ca147bce00dc799947","618c11c2147bce00dc799945"]…

In questo formato i caratteri sono 1448 che corrispondono a 1448 bytes: ciò implica un risparmio di circa il 43% dei dati da inviare.

Eseguendo vari test è stato verificato che questo termine rimane costante anche quando vengono inviate un numero maggiore di misurazioni.

Questa proposta non era mirata a sostituire la *route* attuale per la *get* dei dati, ma si è voluto creare una rotta alternativa più vantaggiosa nel contesto dei big data; questa specifica richiesta permetterebbe di alleviare il peso che si creerebbe sulla *get* attuale in caso di molte richieste in contemporanea di grandi quantità di dati.

Inoltre, la creazione di una *route* parallela permetterebbe di non sconvolgere troppo l'attuale ambiente Measurify, ma di sviluppare nuove possibilità.

Il nuovo formato che è stato deciso di disporre all'utente è quello corrispondente al *Pandas Dataframe* in particolare il *dictionary column oriented*.

	account	Jan	Feb	Mar
0	Jones LLC	150	200	140
1	Alpha Co	200	210	215
2	Blue Inc	50	90	95

Column Oriented

```
sales = {'account': ['Jones LLC', 'Alpha Co', 'Blue Inc'],
         'Jan': [150, 200, 50],
         'Feb': [200, 210, 90],
         'Mar': [140, 215, 95]}
df = pd.DataFrame.from_dict(sales)


from_dict
```

Figura 9: Formato pandas dataframe dictionary column oriented

Per decidere la strada maggiormente ottimizzata per implementare quanto sopra sono stati analizzati vari scenari prima di implementare il codice. Questo può essere riassunto tramite lo schema sotto riportato:

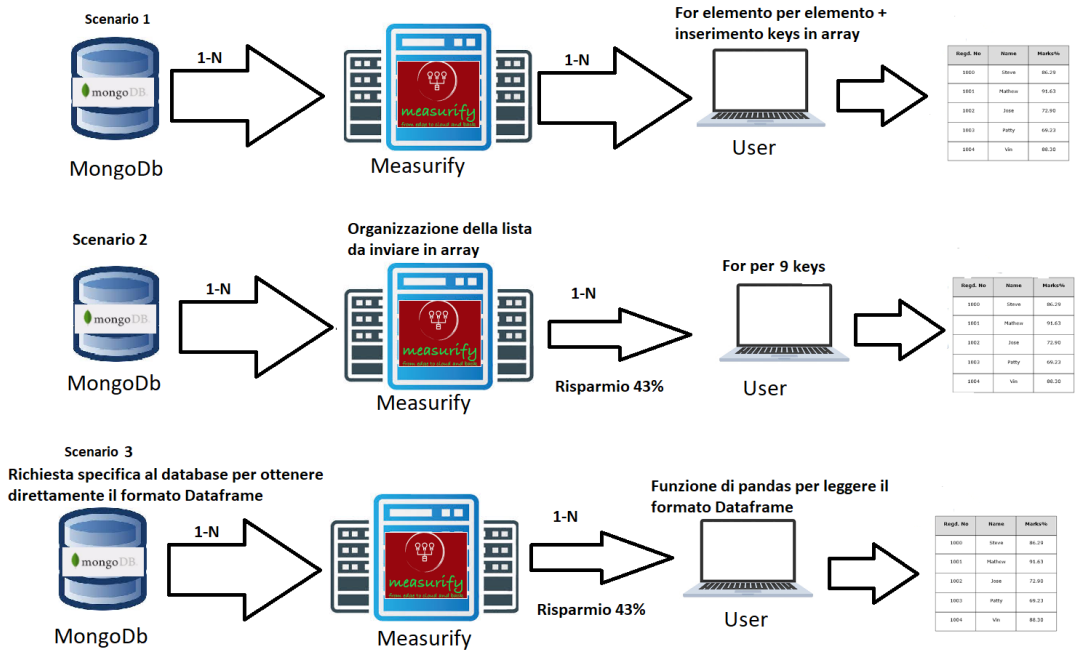

Figura 10: Diversi scenari possibili per l'implementazione

Originariamente l'obiettivo di questa tesi non era presente sulla API e si era deciso di gestire la libreria Python seguendo le specifiche e utilizzando la libreria per organizzare i dati. Si notò subito che questa soluzione era molto dispendiosa in termini di tempo e di memoria in quanto per organizzare tutti i valori in tabella a partire dalle misurazioni in JSON erano presenti dei cicli *for* annidati che gravavano sulla macchina dell'utente, il quale può disporre di scarse risorse.

Il secondo scenario è stato quello di provare a spostare la logica computazionale alla API Measurify in modo che presenti al client i dati già nel formato desiderato, distribuendo il lavoro al server, con prestazioni di gran lunga superiori al PC dell'utente.

Per ultimo, il terzo scenario analizza a fondo le funzionalità di MongoDB: è stata utilizzata la funzione *Aggregation*, la quale modifica la richiesta che faceva la API a MongoDB; questo ha permesso di ottenere i dati già catalogati nel modo richiesto.

Valutando le caratteristiche di ogni scenario si è deciso di utilizzare la terza soluzione in quanto non avrebbe gravato né sull'API né sull'utente finale.

Completata la disponibilità del formato Dataframe si è pensato di aggiungere quello CSV per poter avere uno scambio di informazioni molto più elevato, risparmiando sulla quantità di dati da inviare poiché il CSV stesso è strutturato come una tabella per ridurre al minimo le ridondanze. Piuttosto che eseguire le computazioni con la libreria Python, è stato deciso di implementare queste nuove funzioni direttamente sulla API generando una nuova *route* chiamata *Dataset*. Questa rotta verrà utilizzata per lo scambio di grandi informazioni di dati nei formati CSV, CSV+, Dataframe e Json come descritto precedentemente all'interno del paragrafo getDataset.

Per far ciò, è stato progettato e implementato il codice per la gestione di tutte rotte necessarie nella libreria Python tenendo conto della gestione di tutti i possibili errori e cercando di scrivere codice di alto livello con uno sguardo attento a quanto già

disponibile; una volta finito e testato ciò, sono stati creati gli *unit test* riferiti alle nuove rotte così da assicurarne il funzionamento anche in futuro.

5. Risultati sperimentali

In questa sezione verranno utilizzati diversi dataset per approfondire e testare le funzionalità delle nuove implementazioni sulla API Measurify e libreria measurify.py. Verranno utilizzati diversi dataset provenienti da diversi settori, di cui è stata fatta menzione nel capitolo 2.2 campi di applicazione.

5.1 Dataset per unit testing

Per prima cosa la libreria è stata testata su dataset semplici per verificare il corretto funzionamento delle funzioni.

Il primo dataset utilizzato contiene una singola misurazione:

"2022-02-01T17:54:33.180Z","test-thing-1","test-tag-1","test-device-1","2022-02-01T17:54:33.180Z",10.

Per dare un significato a ogni parte della riga è stato generato il file di *description*:

{

"thing":2,

"device":4,

"items":{

"test-feature":[6]

},

"tags":[3],

"startdate":1,

"enddate":5,

"feature":"test-feature"

}

La misurazione è andata a buon fine in quanto il report di risposta da Measurify equivale a {"completed":["0"],"errors":[]}, mostrando che la riga salvata all'interno del file è stata salvata correttamente.

Il secondo test è stato generato con più misurazioni, di cui la prima è stata scritta correttamente, mentre le altre volutamente presentano tutti gli errori che la API potrebbe riscontrare:

"2022-02-01T17:54:33.180Z","test-thing-1","test-tag-1","test-device-1","2022-02-01T17:54:33.180Z",10,"test-feature"

"2022-01-01T17:54:33.180Z","fake-thing","test-tag-1","test-device-1","2022-01-01T17:54:33.180Z",20,"test-feature"

"2022-03-01T17:54:33.180Z","test-thing-1","test-tag-1","fake-device","2022-03-01T17:54:33.180Z",30,"test-feature"

"2022-03-01T17:54:33.180Z","test-thing-1","test-tag-1","test-device-1","2022-03-01T17:54:33.180Z","test-feature"

"bad2022-04-01T17:54:33.180Z","test-thing-1","test-tag-1","test-device-1","2022-04-01T17:54:33.180Z",40,"test-feature"

"2022-05-01T17:54:33.180Z","test-thing-1","fake-tag","test-device-1","2022-05-01T17:54:33.180Z",50,"test-feature"

"2022-06-01T17:54:33.180Z","test-thing-1","test-tag-1","test-device-1","bad2022-06-01T17:54:33.180Z",60,"test-feature"

"2022-07-01T17:54:33.180Z","test-thing-1","test-tag-1","test-device-1","2022-07-01T17:54:33.180Z",70,"bad-feature"

"2022-08-01T17:54:33.180Z","test-thing-1","test-tag-1","test-device-1","2022-08-01T17:54:33.180Z","a","test-feature"

Il file *description* usato:

```
{
"thing":2,
"device":4,
"items":{
   "test-feature":[6]
},
"tags":[3],
"startdate":1,
"enddate":5,
"feature":7
}
```

La risposta del server a questo caricamento mostra come la prima riga è l'unica caricata correttamente, mentre le altre righe sono presenti nell'array "*errors*" con a fianco una breve spiegazione dell'errore:

```
{
 completed: [ '0' ],
 errors: [
```

'Index: 1 (thing fake-thing not found in database)',

'Index: 2 (device fake-device not found in database)',

'Index: 3 (not enough fields in the row)',

'Index: 4 (startdate is not in Date format)',

'Index: 5 (tag fake-tag not found in database)',

'Index: 6 (enddate is not in Date format)',

'Index: 7 (feature bad-feature not found in database)',

'Index: 8 (expected number in samples at position 0)'

]

}

5.2 Ambito Smart City

Per quanto riguarda l'ambito Smart City è stato utilizzato un dataset che contiene i dati relativi alla qualità dell'aria dell'anno in corso relativi a tre centraline poste nel comune di Bologna (BO). I valori degli inquinanti atmosferici sono espressi in μg/m3. I dati sono stati elaborati partendo da quelli messi a disposizione dall'ARPAE Emilia Romagna.[35]

Come si può vedere dalla seguente tabella, i dati sono rilevati in tempo reale e sono scaricabili tramite file CSV.

	_id	reftime	stazione	value	agente_atm
1	987.792	1 gennaio 2022 00:00	GIARDINI MARGHERITA, BOLOGNA ...	35	PM10
2	988.241	1 gennaio 2022 00:00	GIARDINI MARGHERITA, BOLOGNA ...	2	O3 (Ozono)
3	989.200	1 gennaio 2022 00:00	VIA CHIARINI, BOLOGNA VIA CHIAR...	17	NO2 (Biossido di azoto)
4	990.215	1 gennaio 2022 00:00	PORTA SAN FELICE, BOLOGNA PIAZ...	1	CO (Monossido di carbonio)
5	990.356	1 gennaio 2022 00:00	PORTA SAN FELICE, BOLOGNA PIAZ...	2,3	C6H6 (Benzene)
6	987.793	1 gennaio 2022 00:00	PORTA SAN FELICE, BOLOGNA PIAZ...	36	PM10
7	990.938	1 gennaio 2022 00:00	PORTA SAN FELICE, BOLOGNA PIAZ...	57	NO (Monossido di azoto)
8	991.119	1 gennaio 2022 00:00	GIARDINI MARGHERITA, BOLOGNA ...	27	PM2.5
9	987.796	1 gennaio 2022 00:00	VIA CHIARINI, BOLOGNA VIA CHIAR...	28	PM10
10	989.104	1 gennaio 2022 00:00	GIARDINI MARGHERITA, BOLOGNA ...	20	NO2 (Biossido di azoto)
11	989.128	1 gennaio 2022 00:00	PORTA SAN FELICE, BOLOGNA PIAZ...	37	NO2 (Biossido di azoto)
12	989.951	1 gennaio 2022 00:00	PORTA SAN FELICE, BOLOGNA PIAZ...	124	NOX (Ossidi di azoto)
13	991.120	1 gennaio 2022 00:00	PORTA SAN FELICE, BOLOGNA PIAZ...	28	PM2.5
14	995.018	1 gennaio 2022 00:00	VIA CHIARINI, BOLOGNA VIA CHIAR...	0	O3 (Ozono)
15	994.269	1 gennaio 2022 01:00	PORTA SAN FELICE, BOLOGNA PIAZ...	85	NO (Monossido di azoto)
16	992.448	1 gennaio 2022 01:00	GIARDINI MARGHERITA, BOLOGNA ...	19	NO2 (Biossido di azoto)
17	992.472	1 gennaio 2022 01:00	PORTA SAN FELICE, BOLOGNA PIAZ...	39	NO2 (Biossido di azoto)
18	992.544	1 gennaio 2022 01:00	VIA CHIARINI, BOLOGNA VIA CHIAR...	14	NO2 (Biossido di azoto)
19	993.552	1 gennaio 2022 01:00	PORTA SAN FELICE, BOLOGNA PIAZ...	1,2	CO (Monossido di carbonio)
20	991.601	1 gennaio 2022 01:00	GIARDINI MARGHERITA, BOLOGNA ...	2	O3 (Ozono)

Figura 11: Dataset delle misurazioni.[35]

Il file CSV aperto con blocco note e con qualche piccola modifica si presenta così:

```
987792,2022-01-01T00:00:00+01:00,"GIARDINI MARGHERITA; BOLOGNA VIALE BOTTONELLI",35.0,PM10
988241,2022-01-01T00:00:00+01:00,"GIARDINI MARGHERITA; BOLOGNA VIALE BOTTONELLI",2.0,O3(Ozono)
989200,2022-01-01T00:00:00+01:00,"VIA CHIARINI; BOLOGNA VIA CHIARINI",17.0,NO2(Biossido-di-azoto)
990215,2022-01-01T00:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",1.0,CO(Monossido-di-carbonio)
990356,2022-01-01T00:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",2.3,C6H6(Benzene)
994269,2022-01-01T01:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",85.0,NO(Monossido-di-azoto)
992449,2022-01-01T02:00:00+01:00,"GIARDINI MARGHERITA; BOLOGNA VIALE BOTTONELLI",19.0,NO2(Biossido-di-azoto)
992473,2022-01-01T02:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",37.0,NO2(Biossido-di-azoto)
993553,2022-01-01T02:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",1.2,CO(Monossido-di-carbonio)
993695,2022-01-01T02:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",3.0,C6H6(Benzene)
995020,2022-01-01T02:00:00+01:00,"VIA CHIARINI; BOLOGNA VIA CHIARINI",0.0,O3(Ozono)
991603,2022-01-01T03:00:00+01:00,"GIARDINI MARGHERITA; BOLOGNA VIALE BOTTONELLI",2.0,O3(Ozono)
993554,2022-01-01T03:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",1.5,CO(Monossido-di-carbonio)
993696,2022-01-01T03:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",3.7,C6H6(Benzene)
995021,2022-01-01T03:00:00+01:00,"VIA CHIARINI; BOLOGNA VIA CHIARINI",0.0,O3(Ozono)
992547,2022-01-01T04:00:00+01:00,"VIA CHIARINI; BOLOGNA VIA CHIARINI",16.0,NO2(Biossido-di-azoto)
991605,2022-01-01T05:00:00+01:00,"GIARDINI MARGHERITA; BOLOGNA VIALE BOTTONELLI",2.0,O3(Ozono)
992549,2022-01-01T06:00:00+01:00,"VIA CHIARINI; BOLOGNA VIA CHIARINI",14.0,NO2(Biossido-di-azoto)
993293,2022-01-01T06:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",196.0,NOX(Ossidi-di-azoto)
991607,2022-01-01T07:00:00+01:00,"GIARDINI MARGHERITA; BOLOGNA VIALE BOTTONELLI",2.0,O3(Ozono)
993294,2022-01-01T07:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",249.0,NOX(Ossidi-di-azoto)
993558,2022-01-01T07:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",1.4,CO(Monossido-di-carbonio)
993700,2022-01-01T07:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",3.0,C6H6(Benzene)
995025,2022-01-01T07:00:00+01:00,"VIA CHIARINI; BOLOGNA VIA CHIARINI",0.0,O3(Ozono)
992455,2022-01-01T08:00:00+01:00,"GIARDINI MARGHERITA; BOLOGNA VIALE BOTTONELLI",15.0,NO2(Biossido-di-azoto)
992552,2022-01-01T09:00:00+01:00,"VIA CHIARINI; BOLOGNA VIA CHIARINI",14.0,NO2(Biossido-di-azoto)
993560,2022-01-01T09:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",1.2,CO(Monossido-di-carbonio)
991610,2022-01-01T10:00:00+01:00,"GIARDINI MARGHERITA; BOLOGNA VIALE BOTTONELLI",2.0,O3(Ozono)
991649,2022-01-01T10:00:00+01:00,"VIA CHIARINI; BOLOGNA VIA CHIARINI",0.0,O3(Ozono)
992553,2022-01-01T10:00:00+01:00,"VIA CHIARINI; BOLOGNA VIA CHIARINI",21.0,NO2(Biossido-di-azoto)
993297,2022-01-01T10:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",149.0,NOX(Ossidi-di-azoto)
993561,2022-01-01T10:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",1.1,CO(Monossido-di-carbonio)
994279,2022-01-01T11:00:00+01:00,"PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN FELICE",65.0,NO(Monossido-di-azoto)
```

Figura 12: Dataset delle misurazioni su blocco note

In questo file di testo ogni riga corrisponde a una misurazione singola; per questo esperimento si è deciso di utilizzare oltre 10000 misure per valutare le prestazioni del server.

La principale colonna del dataset è l'ultima, quella relativa all'elemento inquinante: essa contiene la feature di cui ogni riga va a studiarne il valore. Successivamente possiamo vedere come la prima colonna presenta il codice univoco di ogni misurazione mentre la penultima contiene il valore rilevato in μg/m3: questi due termini fanno parte dei "*values*" della feature; perciò ogni feature conterrà due valori numerici, il primo sarà il valore misurato nell'aria, il secondo sarà il codice identificativo della misurazione.

La colonna della data verrà utilizzata per compilare sia *startdate* che *enddate* e la terza colonna indica la *thing* ossia la posizione in cui è situata la centralina.

Non essendo presente una colonna Device si utilizzerà un nome comune a tutte le misurazioni e verrà dunque impostato a "Centralina".

Fatte queste considerazioni si può creare il file in formato JSON che verrà allegato al file delle misurazioni per far comprendere alla API a cosa corrisponde ogni singola porzione della riga.

```
{
"thing":3,
"device":"CENTRALINA",
"items":{
"PM10":[4,1],
"O3(Ozono)":[4,1],
"NO2(Biossido-di-azoto)":[4,1],
"CO(Monossido-di-carbonio)":[4,1],
"C6H6(Benzene)":[4,1],
"NO(Monossido-di-azoto)":[4,1],
"NOX(Ossidi-di-azoto)":[4,1],
"PM2.5":[4,1]
},
"tags":[],
"startdate":2,
"enddate":2,
"feature":5
}
```


Figura 13: File description utilizzato per il dataset di Bologna

Come si vede per le chiavi thing, startdate, enddate e feature, sono stati indicati i numeri della colonna corrispondente, mentre per il device è stato messo il nome comune a tutte le misurazioni; per quanto riguarda il campo items, questo è quello che presenta la maggiore complessità in quanto contiene tutte le feature della colonna 5 presenti nel dataset e per ognuna è specificato in quale colonna si trovano i valori. (N.B. in questo caso i valori delle feature si trovano sempre nei campi 4 e 1, ma è possibile anche avere features con un diverso numero di campi e in colonne differenti).

Un esempio per vedere come utilizzare la libreria è il seguente:

```
import measurify
api=measurify.Create()
response=api.postDataset(file="…centraline-qualita-aria.csv",description="…Bologna-description.txt",force=True,verbose=True)
```

```
filename="centraline-qualita-aria"

filter=api.createFilter(feature="PM10")

dataset=api.getDataset(filter=filter,filename=filename,format=api.Format.DATAFRAME
)
print(dataset)
```

Per prima cosa viene inizializzata la libreria e viene eseguita la POST di tutti i valori presenti nel dataset mettendo la variabile Force=True per evitare di inserire manualmente ogni Thing e Device. Una volta completata, si è scelto di impostare il filtro per ricavare solo i valori inquinanti di PM10; successivamente, è stata eseguita una *get* tenendo conto del filtro, del dataset da cui prendere i valori e del formato *Dataframe*.

Per quanto riguarda la risposta alla post, si osservi come tutte le 10057 righe siano state salvate correttamente senza alcun errore:

{"completed":["0","1","2","3","4","5",…,"10056","10057"],"errors":[]}

La risposta alla getDataset è in formato *Pandas Dataframe*, la cui *print* mostra tutte le colonne relativa al filtraggio PM10. Come si vede dall'immagine in basso, delle 10057 righe solo 140 sono quelle relative a PM10.

	_id	visibility	tags	...	thing	device	samples
0	PM10	private	[centraline-qualita-aria]	...	GIARDINI MARGHERITA; BOLOGNA VIALE BOTTONELLI	CENTRALINA	[35, 987792]
1	PM10	private	[centraline-qualita-aria]	...	VIA CHIARINI; BOLOGNA VIA CHIARINI	CENTRALINA	[23, 994543]
2	PM10	private	[centraline-qualita-aria]	...	PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN ...	CENTRALINA	[23, 1001893]
3	PM10	private	[centraline-qualita-aria]	...	VIA CHIARINI; BOLOGNA VIA CHIARINI	CENTRALINA	[12, 1001895]
4	PM10	private	[centraline-qualita-aria]	...	GIARDINI MARGHERITA; BOLOGNA VIALE BOTTONELLI	CENTRALINA	[5, 1008943]
..	...	...	...	...	...	...	...
136	PM10	private	[centraline-qualita-aria]	...	VIA CHIARINI; BOLOGNA VIA CHIARINI	CENTRALINA	[14, 1144531]
137	PM10	private	[centraline-qualita-aria]	...	VIA CHIARINI; BOLOGNA VIA CHIARINI	CENTRALINA	[28, 1148531]
138	PM10	private	[centraline-qualita-aria]	...	GIARDINI MARGHERITA; BOLOGNA VIALE BOTTONELLI	CENTRALINA	[31, 1152107]
139	PM10	private	[centraline-qualita-aria]	...	PORTA SAN FELICE; BOLOGNA PIAZZA DI PORTA SAN ...	CENTRALINA	[30, 1152108]
140	PM10	private	[centraline-qualita-aria]	...	VIA CHIARINI; BOLOGNA VIA CHIARINI	CENTRALINA	[28, 1152111]

Figura 14: Risposta alla get in formato Dataframe

Ottenute questi valori l'utente ne può disporre liberamente per le fasi di analisi, predizione o per il machine learning.
Un esempio di utilizzo di questi dati è disponibile sul sito da cui è stato estratto il dataset: esso viene presentato sotto forma di grafico, nel quale sono state tracciate le medie dei valori presi in esame.

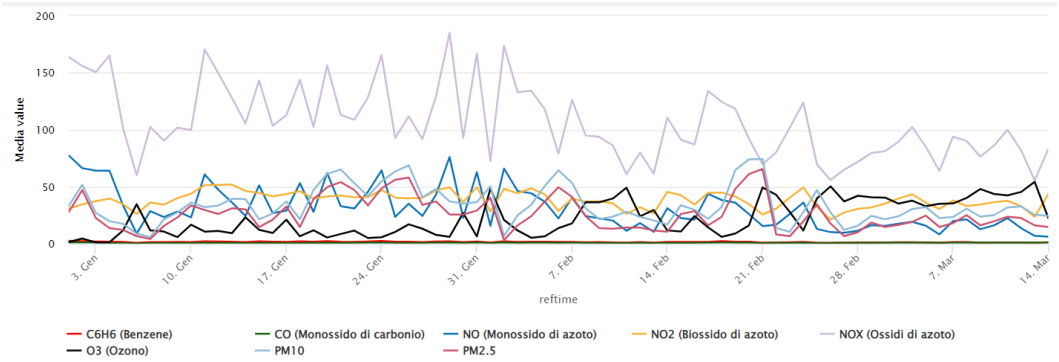

Figura 15: Grafico delle misurazioni di agenti inquinanti nell'aria

5.3 Ambito biomedicale

Il secondo dataset preso in considerazione è relativo all'ambito biomedicale che tiene conto delle misurazioni cardiache di diversi pazienti, con l'aggiunta di alcuni dati anagrafici. Il device da cui sono presi i dati è un cardiofrequenzimetro e ogni colonna ha il suo interesse specifico in campo medico.

È stato scelto questo dataset perché a differenza di quello della Smart City presenta solo valori e non ha componenti al suo interno come Thing, Device, StartDate ecc. Questo dimostra come il codice implementato sia flessibile per coprire la maggior parte di dataset esistenti; si noti come qui non sia presente neanche una colonna che contenga il codice identificativo univoco della linea e pertanto si vedrà che nel caso di due righe perfettamente identiche, il server non salverà la seconda in quanto già presente sul database. Ciò non dovrebbe avvenire nel momento in cui ogni riga ha

un identificatore ma è stato scelto questo dataset come caso limite per vedere il comportamento della API.

I valori del file heart.csv sono i seguenti:

```
age,sex,cp,trestbps,chol,fbs,restecg,thalach,exang,oldpeak,slope,ca,thal,target
63,1,3,145,233,1,0,150,0,2.3,0,0,1,1
37,1,2,130,250,0,1,187,0,3.5,0,0,2,1
41,0,1,130,204,0,0,172,0,1.4,2,0,2,1
56,1,1,120,236,0,1,178,0,0.8,2,0,2,1
57,0,0,120,354,0,1,163,1,0.6,2,0,2,1
57,1,0,140,192,0,1,148,0,0.4,1,0,1,1
56,0,1,140,294,0,0,153,0,1.3,1,0,2,1
44,1,1,120,263,0,1,173,0,0,2,0,3,1
52,1,2,172,199,1,1,162,0,0.5,2,0,3,1
57,1,2,150,168,0,1,174,0,1.6,2,0,2,1
54,1,0,140,239,0,1,160,0,1.2,2,0,2,1
48,0,2,130,275,0,1,139,0,0.2,2,0,2,1
49,1,1,130,266,0,1,171,0,0.6,2,0,2,1
64,1,3,110,211,0,0,144,1,1.8,1,0,2,1
58,0,3,150,283,1,0,162,0,1,2,0,2,1
50,0,2,120,219,0,1,158,0,1.6,1,0,2,1
58,0,2,120,340,0,1,172,0,0,2,0,2,1
66,0,3,150,226,0,1,114,0,2.6,0,0,2,1
43,1,0,150,247,0,1,171,0,1.5,2,0,2,1
69,0,3,140,239,0,1,151,0,1.8,2,2,2,1
59,1,0,135,234,0,1,161,0,0.5,1,0,3,1
44,1,2,130,233,0,1,179,1,0.4,2,0,2,1
42,1,0,140,226,0,1,178,0,0,2,0,2,1
```

Figura 16: Dataset delle misurazioni

L'allegato per la descrizione che si è deciso di utilizzare inserisce i valori di tutti i parametri mancanti:

```
{
"thing":"Ospedale",
"device":"cardiofrequenzimetro",
"items":{
"heartValues":[1,2,3,4,5,6,7,8,9,10,11,12,13,14]
},
"tags":[],
"startdate":"2022-01-01T00:00:00+01:00",
"enddate":"2022-01-01T00:00:00+01:00",
"feature":"heartValues"
}
```


Figura 17: File description utilizzato

Seguendo lo stesso codice Python utilizzato per il precedente dataset, la funzione PostDataset ha caricato correttamente tutti i valori con due eccezioni:

{"completed":["1","2","3","4","5",…,"302","303"],"errors":["Index: 0 (expected number in samples at position 0)","Index: 165 (Error: The measurement already exists)"]}

Da questo report vediamo come 302 sono state caricate correttamente.

La prima riga non è stata salvata in quanto conteneva l'*header* del CSV, che nell'approccio di Measurify, la sua funzionalità è stata sostituita dal file di descrizione.

inoltre, la riga 165 non è stata inserita perché la misura esiste già e possiamo andare a constatare che questa è identica a quella precedente:

```
38,1,2,138,175,0,1,173,0,0,2,4,2,1
38,1,2,138,175,0,1,173,0,0,2,4,2,1
```

Per motivi di idempotenza è stato deciso di richiedere all'utente che il file CSV abbia una colonna identificativa per essere certi che ogni riga sia differente oppure di accettare che le righe uguali non vengano salvate.

Una volta caricati i valori questi possono essere ricavati per esempio in formato CSV, usando la getDataset:

```
filename="heart"
filter=api.createFilter(feature="heartValues")
dataset=api.getDataset(filter=filter,filename=filename,format=api.Format.CSV)
print(dataset)
```

Come risultato della *print* otteniamo tutto il dataset più i parametri aggiunti nel file di descrizione.

```
"2021-12-31T23:00:00.000Z","Ospedale","heartValues","cardiofrequenzimetro","[57,1,0,140,192,0,1,148,0,0.4,1,0,1,1]",0
"2021-12-31T23:00:00.000Z","Ospedale","heartValues","cardiofrequenzimetro","[57,0,0,120,354,0,1,163,1,0.6,2,0,2,1]",0
"2021-12-31T23:00:00.000Z","Ospedale","heartValues","cardiofrequenzimetro","[56,1,1,120,236,0,1,178,0,0.8,2,0,2,1]",0
"2021-12-31T23:00:00.000Z","Ospedale","heartValues","cardiofrequenzimetro","[41,0,1,130,204,0,0,172,0,1.4,2,0,2,1]",0
"2021-12-31T23:00:00.000Z","Ospedale","heartValues","cardiofrequenzimetro","[37,1,2,130,250,0,1,187,0,3.5,0,0,2,1]",0
"2021-12-31T23:00:00.000Z","Ospedale","heartValues","cardiofrequenzimetro","[63,1,3,145,233,1,0,150,0,2.3,0,0,1,1]",0'
```

Figura 18: Risposta alla Get in formato CSV

Questo dimostra come è possibile caricare dataset che non presentino tutte le colonne necessarie al modello di misurazioni di Measurify; questo discorso può essere estremizzato fino ad arrivare a dataset puramente numerici utilizzando il corretto file di descrizione da inviare alla API.

5.4 Ambito Domotica e Industria

Il terzo dataset copre l'ambito della domotica e può essere utile anche per monitoraggi a scopo industriale; esso contiene le misurazioni ottenute da sensori presenti in una stanza quali la temperatura, umidità e la quantità di luce. Oltre questi valori sono stati testati differenti scenari come lo stato aperto-chiuso di una porta e di una finestra oppure aggiungendo persone all'interno di una stanza e monitorando il loro comportamento.[36]

Il file CSV contiene le seguenti informazioni:

<EID>, <AbsT>, <RelT>, <NID>, <Temp>, <RelH>, <L1>, <L2>, <Occ>, <Act>, <Door>, <Win>

I campi sopra elencati hanno il seguente significato:

<EID>: ID ingresso

<AbsT>: Timestamp assoluto [ms]

<RelT>: Timestamp relativo [s]

<NID>: ID del nodo

Dati del sensore:

<Temp>: Temperatura [°C]

<RelH>: Umidità relativa [%]

<L1>: Sensore di luce 1 (lunghezza d'onda) [nm]

<L2>: Sensore di luce 2 (lunghezza d'onda) [nm]

Stato della stanza:

<Occ>: Numero di occupanti (0, 1, 2)

<Act>: Attività degli occupanti (0 = n/a, 1 = leggere, 2 = stare in piedi, 3 = camminare, 4 = lavorare)

<Door>: Stato della porta (0 = chiuso, 1 = aperto)

<Win>: Stato della finestra (0 = chiuso, 1 = aperto)

Per questa sperimentazione è stato scelto il primo dataset della stanza A, contenente 8183 misurazioni; verranno impostate la thing come *Room_A* e device come *Room_climate*.

Nella foto seguente sono presenti i valori delle prime 25 misurazioni.

```
1, 1458031648545, 1, 1, 20.48, 42.332, 185.71, 492.2, 0, 0, 0, 0
2, 1458031648645, 1, 4, 20.73, 39.983, 214.29, 657.8, 0, 0, 0, 0
3, 1458031648795, 1, 3, 20.48, 38.687, 197.14, 542.8, 0, 0, 0, 0
4, 1458031651038, 3, 2, 20.89, 40.712, 174.29, 552.0, 0, 0, 0, 0
5, 1458031652010, 4, 4, 20.73, 39.983, 211.43, 653.2, 0, 0, 0, 0
6, 1458031652409, 4, 3, 20.48, 38.687, 198.57, 542.8, 0, 0, 0, 0
7, 1458031652680, 5, 1, 20.48, 42.332, 185.71, 496.8, 0, 0, 0, 0
8, 1458031655028, 7, 2, 20.90, 40.712, 175.71, 547.4, 0, 0, 0, 0
9, 1458031656375, 8, 4, 20.73, 39.983, 214.29, 657.8, 0, 0, 0, 0
10, 1458031656640, 9, 3, 20.47, 38.687, 198.57, 533.6, 0, 0, 0, 0
11, 1458031656668, 9, 1, 20.48, 42.332, 185.71, 492.2, 0, 0, 0, 0
12, 1458031659143, 11, 2, 20.90, 40.712, 171.43, 542.8, 0, 0, 0, 0
13, 1458031660131, 12, 1, 20.48, 42.332, 187.14, 501.4, 0, 0, 0, 0
14, 1458031660620, 13, 3, 20.47, 38.687, 197.14, 542.8, 0, 0, 0, 0
15, 1458031661115, 13, 4, 20.74, 39.983, 211.43, 653.2, 0, 0, 0, 0
16, 1458031662633, 15, 2, 20.90, 40.712, 171.43, 542.8, 0, 0, 0, 0
17, 1458031664480, 16, 4, 20.74, 39.983, 211.43, 653.2, 0, 0, 0, 0
18, 1458031664735, 17, 3, 20.48, 38.687, 197.14, 547.4, 0, 0, 0, 0
19, 1458031664774, 17, 1, 20.49, 42.332, 187.14, 492.2, 0, 0, 0, 0
20, 1458031667123, 19, 2, 20.90, 40.712, 171.43, 542.8, 0, 0, 0, 0
21, 1458031667985, 20, 1, 20.48, 42.332, 187.14, 492.2, 0, 0, 0, 0
22, 1458031668720, 21, 4, 20.74, 39.983, 211.43, 653.2, 0, 0, 0, 0
23, 1458031669090, 21, 3, 20.47, 38.687, 197.14, 533.6, 0, 0, 0, 0
24, 1458031670988, 23, 2, 20.89, 40.712, 172.86, 547.4, 0, 0, 0, 0
25, 1458031672069, 24, 3, 20.48, 38.687, 197.14, 547.4, 0, 0, 0, 0
```

Figura 19: Dataset delle misurazioni

Secondo quanto detto in precedenza, l'allegato per la descrizione che si è deciso di utilizzare conterrà i valori di tutti i parametri mancanti:

```
{
"thing":"Room_A",
"device":"Room_climate",
"items":{
"Room_Status":[1,3,4,5,6,7,8,9,10,11,12]
},
"tags":[],
"startdate":2,
"enddate":2,
"feature":"Room_Status"
}
```


Figura 20: File description utilizzato

Seguendo lo stesso codice Python utilizzato per il primo dataset per caricare tutte le misurazioni e poterne disporne direttamente dal database, è stata utilizzata la funzione PostDataset. In figura possiamo vedere il report della funzione che indica come tutte le misurazioni sono state caricate correttamente:

{"completed":["0","1","2","3","4","5",…,"8182","8183"],"errors":[]}

Per ottenere i valori e controllare l'esito, si può utilizzare la *getDataset*, e poiché la feature è fissata, si può procedere con la scelta del formato CSV+ da cui si ottiene:


```
"device","EID","RelT","NID","Temp","RelH","L1","L2","Occ","Act","Door","Win"
"Room_climate","8184","8177","1","20.72","45.45","200","575","0","0","0","0","0"
"Room_climate","8183","8176","4","21.13","42.615","232.86","791.2","0","0","0","0"
"Room_climate","8182","8176","3","20.75","41.481","215.71","680.8","0","0","0","0"
"Room_climate","8181","8174","2","21.12","43.79","188.57","602.6","0","0","0","0"
"Room_climate","8180","8173","3","20.74","41.522","215.71","680.8","0","0","0","0"
"Room_climate","8179","8172","1","20.72","45.45","201.43","584.2","0","0","0","0"
"Room_climate","8178","8172","4","21.13","42.575","232.86","791.2","0","0","0","0"
"Room_climate","8177","8171","2","21.12","43.79","188.57","602.6","0","0","0","0"
"Room_climate","8176","8169","3","20.74","41.522","217.14","676.2","0","0","0","0"
"Room_climate","8175","8169","4","21.14","42.575","235.71","805","0","0","0","0"
```


Figura 21: alcuni campi del CSV+ ottenuto come risposta alla Get

Nel formato CSV+ non è contenuto un campo generico *values*, ma è presente la suddivisione in tante colonne quanti sono i parametri della feature. Grazie alla possibilità di aggiungere diversi filtri, le possibilità di analisi a partire da questo dataset sono numerose, come per esempio studiare la variazione dei valori all'apertura della porta o della finestra, oppure come la presenza di persone nella stanza modifichi l'aria all'interno di essa.

6. Conclusioni e lavori futuri

L'ambiente IoT si sta rivelando uno dei maggiori settori in rapida espansione sul mercato. Le potenzialità offerte da questa architettura risultano al passo coi tempi in un mondo in cui sempre più dispositivi diversi tra loro richiedono la connessione alla rete con vincoli di bassa latenza e alto livello di sicurezza.

La tesi ha esteso Measurify, framework open source orientato alla gestione delle misure, che sfrutta lo stato dell'arte delle tecnologie di Big Data Management, supportando uno sviluppo di applicazioni efficace ed efficiente.

Il presente elaborato ha presentato l'estensione delle funzionalità di Measurify per il caricamento e scaricamento efficiente di interi dataset, minimizzando la necessità di intervento da parte dello sviluppatore/data scientist.

La tesi ha realizzato anche una libreria python che permette di accedere alle principali funzionalità di Measurify utilizzando il linguaggio di riferimento per i data scientists per l'analisi di dati, e lo sviluppo e l'impiego di modelli di machine learning; si è cercato di fornire all'utente una libreria molto flessibile, in grado di adattarsi a dataset provenienti da vari settori d'applicazione.

Come mostrato nel capitolo sulla sperimentazione, Measurify è ora in grado di gestire, anche dal punto di vista del versionamento, una notevole varietà di dati in formato CSV proveniente da numerosi settori IoT. I casi di studio proposti hanno riguardato: dati provenienti da dispositivi di un Comune italiano per il monitoraggio delle condizioni atmosferiche; dati d'ambiente, quali la temperatura e l'umidità dell'aria all'interno di una stanza, per scopi industriali e di ricerca; dati provenienti da apparecchiature mediche. Tutti i test hanno coinvolto dataset con grosse quantità e varia strutturazione di dati, pur nel comune formato CSV, e i risultati presentati dimostrano la validità del livello di astrazione del progetto delle API, che permette all'utente di caricare e scaricare dataset organizzati in maniera anche molto diversa

tra loro (ad esempio, feature multidimensionli, con/senza tag, serie temporali o campioni indipendenti ed identicamente distribuiti, etc.), semplicemente scrivendo poche righe di configurazione in formato JSON.

Il progetto presentato ha realizzato un passo significativo per l'estensione di Measurify nella direzione del supporto dello sviluppo di modelli di intelligenza artificiale e machine learning. Le prossime estensioni sono previste per il supporto di dataset di immagini e video e per la gestione dei modelli stessi di machine learning.

Riferimenti bibliografici e sitografici

[1] https://it.wikipedia.org/wiki/Application_programming_interface

[2] https://measurify.org/

[3] https://en.wikipedia.org/wiki/Internet_of_things

[4] https://it.wikipedia.org/wiki/Representational_state_transfer

[5] https://www.mongodb.com/

[6] https://nodejs.org/it/

[7] https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/

[8] https://techjury.net/blog/how-many-iot-devices-are-there/

[9] https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/iot-value-set-to-accelerate-through-2030-where-and-how-to-capture-it

[10] https://www.i-scoop.eu/internet-of-things-iot/internet-robotic-things-iort/

[11] https://www.internet4things.it/iot-library/lavionica-verso-liot/

[12] https://www.cybersecurity360.it/cybersecurity-nazionale/internet-of-battlefield-things-liot-nel-contesto-militare-scenari-e-soluzioni-di-cyber-security/

[13] https://www.internet4things.it/edge-computing/analytics-big-data/big-data-iot-cosa-sono-e-vantaggi-dellanalisi-per-le-aziende/

[14] https://ieeexplore.ieee.org/document/9093133

[15] S. Cheruvu, A. Kumar, N. Smith, and D. M. Wheeler, “IoT frameworks and complexity,” in Proc. Demystifying Internet Things Secur., 2020, pp. 23–148.

[16] A. Atmani, I. Kandrouch, N. Hmina, and H. Chaoui, “Big data for internet of things: A survey on iot frameworks and platforms,” in Advanced Intelligent Systems for Sustainable Development (Lecture Notes in Netw. Syst.), vol. 92. M. Ezziyyani,Ed. Cham,The Netherelands: Springer, 2020. [Online]. Available: https://doi.org/10.1007/978-3-030-33103-0_7

[17] Amazon Web Services AWS IoT, 2019. [Online]. Available: https://aws. amazon.com/iot/solutions/industrial-iot/?nc = sn&loc = 3&dn = 2

[18] W. Tarneberg, V. Chandrasekaran, and M. Humphrey, “Experiences creating a framework for smart traffic control using AWS IOT,” in Proc. 9th Int. Conf. Utility Cloud Comput., New York, NY, USA, 2016, pp. 63–69.

[19] Azure IoT, Microsoft, Redmond, WA, USA, 2019. [Online]. Available: https://azure.microsoft.com/en-us/overview/iot/

[20] S. Jiong, J. Liping, and L. Jun, “The integration of azure sphere and azure cloud services for internet of things,” MDPI J. Appl. Sci., vol. 9, no. 13, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/13/2746

[21] T. Pflanzner and A. Kertesz, “A survey of IoT cloud providers,” in Proc. 39th Int. Conv. Inf. Commun. Technol., Electron. Microelectron., 2016, pp. 730–735.

[22] M. Zúñiga-Prieto, J. González-Huerta, E. Insfran, and S. Abrahão, “Dynamic reconfiguration of cloud application architectures,” J. Softw., Pract.

Experience, vol. 48, pp. 327–344, 2016.

[23] H. Chi, T. Aderibigbe, and B. C. Granville, “A framework for IoT data acquisition and forensics analysis,” in Proc. IEEE Int. Conf. Big Data, Seattle, WA, USA, 2018, pp. 5142–5146.

[24] J. Jung, K. Kim, and J. Park, “Framework of big data analysis about IoT-Home-device for supporting a decision making an effective strategy about new product design,” in Proc. Int. Conf. Artif. Intell. Inf. Commun., Okinawa, Japan, 2019, pp. 582–584.

[25] ¸ S. Kolozali et al., “Observing the pulse of a city: A smart city framework for real-time discovery, federation, and aggregation of data streams,” IEEE Internet Things J., vol. 6, no. 2, pp. 2651–2668, Apr. 2019.

[26] L. Jiang, L. D. Xu, H. Cai, Z. Jiang, F. Bu, and B. Xu, “An IoT-oriented data storage framework in cloud computing platform,” IEEE Trans. Ind. Informat., vol. 10, no. 2, pp. 1443–1451, May 2014.

[27] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, “IoT-based big data storage systems in cloud computing: perspectives and challenges,” IEEE Internet Things J., vol. 4, no. 1, pp. 75–87, Feb. 2017.

[28] J. Fu, Y. Liu, H. Chao, B. K. Bhargava, and Z. Zhang, “Secure data storage and searching for industrial IoT by integrating fog computing and cloud computing,” IEEE Trans. Ind. Informat., vol. 14, no. 10, pp. 4519–4528,

Oct. 2018.

[29] F. Paganelli, S. Turchi, and D. Giuli, “A web of things framework for RESTful applications and its experimentation in a smart city,” IEEE Syst. J., vol. 10, no. 4, pp. 1412–1423, Dec. 2016.

[30] S. K. Sharma and X. Wang, “Live data analytics with collaborative edge and cloud processing in wireless IoT networks,” IEEE Access, vol. 5, pp. 4621–4635, 2017. [Online]. Available: https://doi.org/10.1109/ACCESS.2017.2682640

[31] T. Yu, X.Wang, and A. Shami, “Recursive principal component analysisbased data outlier detection and sensor data aggregation in IoT systems,” IEEE Internet Things J., vol. 4, no. 6, pp. 2207–2216, Dec. 2017.

[32] https://spark.apache.org/

[33] https://github.com/measurify/dataset

[34] https://github.com/measurify

[35] https://opendata.comune.bologna.it/explore/dataset/centraline-qualita-aria/information/?disjunctive.agente_atm

[36] https://github.com/IoTsec/Room-Climate-Datasets

Ringraziamenti

Vorrei ringraziare il Professor Riccardo Berta e il Professor Francesco Bellotti per avermi aiutato in questo lavoro di tesi.

Ringrazio i Dottorandi Alessio Capello, Marianna Cossu e Luca Lazzaroni per la disponibilità e la cordialità dimostrate nel corso di questi mesi.

Grazie agli amici più cari, una certezza, e ai compagni di corso, per merito dei quali questi cinque anni sono volati via leggeri.

Il ringraziamento più grande va infine alla mia famiglia, che mi ha sempre supportato e spronato a fare di più.

Matteo