Destinatari

Prof. Tullio Vardanega
Prof. Riccardo Cardin

Redattori

Davide Donanzan
Pietro Busato

Verificatori

Guglielmo Barison
Veronica Tecchiati
Linda Barbiero

Specifica Tecnica

nan1fyteam.unipd@gmail.com

Registro delle Modifiche

Versione	Data	Descrizione	Redattore	Verificatore
1.0.0	2024-08-18	**Approvazione per PB**		
0.6.0	2024-08-16	Completamento sezione 3 e rifiniture.	Davide Donanzan	Linda Barbiero
0.5.0	2024-08-15	Continuazione sezione 3.	Pietro Busato	Linda Barbiero
0.4.0	2024-08-12	Stesura sezione 3 e 4.	Pietro Busato	Veronica Tecchiati
0.3.0	2024-08-10	Completamento sezione 5.	Davide Donanzan	Pietro Busato
0.2.0	2024-08-09	Continuazione sezione 2 e stesura sezione 5.	Davide Donanzan	Pietro Busato
0.1.0	2024-07-25	Stesura sezione 2.	Davide Donanzan	Guglielmo Barison
0.0.0	2024-07-01	Stesura struttura e sezione 1.	Davide Donanzan	Guglielmo Barison

Tabella: Registro delle modifiche.

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Il presente documento si propone come risorsa esaustiva per la comprensione degli aspetti tecnici chiave del progetto SyncCity. La sua finalità primaria è fornire una descrizione dettagliata e approfondita dell'architettura[G] implementativa del sistema[G]. In particolare, si prevede un'analisi approfondita che si estenda anche al livello di design più basso, includendo definizione e spiegazione dettagliata dei design pattern[G] utilizzati.
L'obiettivo principale del presente documento è triplice: motivare le scelte di sviluppo adottate; fungere da guida fondamentale per l'attività di codifica e di manutenzione; monitorare la copertura dei requisiti identificati nel documento *Analisi dei Requisiti*.
L'adeguatezza del documento e dell'architettura viene costantemente monitorata e modificata sulla base dei requisiti e dei feedback ricevuti da parte della Proponente[G].

1.2 Scopo del prodotto

L'obiettivo del progetto SyncCity è quello di creare una piattaforma atta al monitoraggio di sensori sparsi geograficamente nel territorio di una città. I sensori in questione permettono la misurazione e segnalazione di dati real-time[G] riguardanti le più disparate caratteristiche e necessità del territorio quali temperatura ed umidità esterna, occupazione di stalli di parcheggio, funzionamento o guasto elettrico di colonnine di ricarica, traffico stradale e via dicendo. La Proponente richiede la simulazione di alcuni dei sensori nominati nonchè la gestione dei dati, della loro persistenza e della loro rappresentazione grafica attraverso widget[G] e grafici.

SyncCity permetterà un miglioramento della qualità[G] dei servizi offerti dalla città attraverso il continuo monitoraggio della stessa, ottenendo, gestendo e successivamente condividendo i dati con gli utenti.

1.3 Glossario

Per garantire chiarezza nel linguaggio utilizzato nei documenti, è stato redatto un Glossario contenente le definizioni dei termini con significato specifico da disambiguare. Tali termini sono contrassegnati con una G ad apice. L'inserimento di un termine nel Glossario è considerato completo solo dopo averne fornito la definizione.

1.4 Riferimenti

1.4.1 Normativi

- *Norme di Progetto v2.0.0*;
- Presentazione e documentazione del capitolato[G] d'appalto C6 - SyncCity:
  - https://www.math.unipd.it~tullio/IS-1/2023/Progetto/C6p.pdf (Ultimo accesso: 2024-08-18)
  - https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C6.pdf (Ultimo accesso: 2024-08-18)
- Regolamento di progetto:
  - https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/PD2.pdf (Ultimo accesso: 2024-08-18)

1.4.2 Informativi

- *Analisi dei Requisiti v2.0.0*;
- *Glossario v2.0.0*;
- Diagrammi delle classi (UML) - corso di Ingegneria del Software a.a. 2023/2024:
  - https://www.math.unipd.it/~rcardin/swea/2023/Diagrammi%20delle%20Classi.pdf (Ultimo accesso: 2024-08-18)
- Diagrammi di Sequenza (UML) - corso di Ingegneria del Software a.a. 2023/2024:
  - https://www.math.unipd.it/~rcardin/swea/2022/Diagrammi%20di%20Sequenza.pdf (Ultimo accesso: 2024-08-18)
- Progettazione - corso di Ingegneria del Software a.a. 2023/2024:
  - https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T6.pdf (Ultimo accesso: 2024-08-18)

1.4.3 Tecnici

- Documentazione Docker[G]:
  - https://docs.docker.com/ (Ultimo accesso: 2024-08-18)
- Documentazione Docker Compose[G]:
  - https://docs.docker.com/compose/ (Ultimo accesso: 2024-08-18)
- Documentazione Python[G]:
  - https://docs.python.org/3/ (Ultimo accesso: 2024-08-18)
- Documentazione pytest - Python[G]:
  - https://docs.pytest.org/en/7.1.x/contents.html (Ultimo accesso: 2024-08-18)
- Documentazione unittest - Python[G]:
  - https://docs.python.org/3/library/unittest.html (Ultimo accesso: 2024-08-18)
- Documentazione numpy - Python[G]:
  - https://numpy.org/devdocs/user/ (Ultimo accesso: 2024-08-18)
- Documentazione confluent-kafka - Python[G]:
  - https://developer.confluent.io/get-started/python/ (Ultimo accesso: 2024-08-18)
- Documentazione clickhouse-connect - Python[G]:
  - https://clickhouse.com/docs/en/integrations/python (Ultimo accesso: 2024-08-18)
- Documentazione Apache Kafka[G]:
  - https://kafka.apache.org/20/documentation.html (Ultimo accesso: 2024-08-18)
- Documentazione Apache Flink[G]:
  - https://nightlies.apache.org/flink/flink-docs-stable/ (Ultimo accesso: 2024-08-18)
- Documentazione Java[G]:
  - https://docs.oracle.com/en/java/ (Ultimo accesso: 2024-08-18)
- Documentazione Apache Maven[G]:
  - https://maven.apache.org/guides/index.html (Ultimo accesso: 2024-08-18)

- Documentazione ClickHouse[G]:
  - https://clickhouse.com/docs/en/intro (Ultimo accesso: 2024-08-18)
- Documentazione Kafka Engine - ClickHouse[G]:
  - https://clickhouse.com/docs/en/engines/table-engines/integrations/kafka (Ultimo accesso: 2024-08-18)
- Documentazione materialized view - ClickHouse[G]:
  - https://clickhouse.com/docs/en/guides/developer/cascading-materialized-views (Ultimo accesso: 2024-08-18)
- Documentazione MergeTree - ClickHouse[G]:
  - https://clickhouse.com/docs/en/engines/table-engines/mergetree-family/mergetree#mergetree (Ultimo accesso: 2024-08-18)
- Documentazione Grafana[G]:
  - https://grafana.com/docs/grafana/latest (Ultimo accesso: 2024-08-18)
- Documentazione ClickHouse datasource - Grafana[G]:
  - https://grafana.com/grafana/plugins/grafana-clickhouse-datasource/ (Ultimo accesso: 2024-08-18)
- Documentazione variables - Grafana[G]:
  - https://grafana.com/docs/grafana/latest/variables/ (Ultimo accesso: 2024-08-18)
- Documentazione alerts - Grafana[G]:
  - https://grafana.com/docs/grafana/latest/alerting/ (Ultimo accesso: 2024-08-18)
- Documentazione notification policies - Grafana[G]:
  - https://grafana.com/docs/grafana/latest/alerting-rules/create-notification-policy/ (Ultimo accesso: 2024-08-18)

2 Tecnologie

In questa sezione verranno illustrati gli strumenti e le tecnologie impiegati per lo sviluppo del software nonché infrastrutture, linguaggi di programmazione, librerie[G] e framework[G] utilizzati.

2.1 Docker[G]

In fase di sviluppo, testing e produzione, sono stati utilizzati container[G] Docker[G] per creare ambienti di lavoro efficaci e consistenti.

2.1.1 Ambienti

Sono stati creati quattro ambienti di lavoro con focus operativi differenti.

- **Ambiente di produzione:**
  - Ambiente dove i programmatori possono testare il codice nella produzione dei dati simulati;
  - Il focus è quello della produzione dei dati simulati nell'ottica della fruizione del servizio fornito dall'applicativo;
  - L'intera data pipeline[G], ad esclusione dello stream-processing[G], è in funzione.
- **Ambiente di development:**
  - Ambiente dove i programmatori possono controllare il corretto collegamento tra le tecnologie implementate;
  - Non è attiva la generazione dei dati simulati;
  - Il focus è l'interazione fra le tecnologie della data pipeline[G].
- **Ambiente di test:**
  - Ambiente che simula l'ambiente di produzione;
  - Utilizza test automatizzati per verificare la correttezza del codice nelle sue unità e nell'integrazione di queste tra loro.
- **Ambiente di stream-processing:**
  - Ambiente dove i programmatori possono controllare il corretto funzionamento dello stream-processing[G];
  - Il focus è quello di osservare il comportamento, implementare e verificare l'elaborazione di stream differenti;
  - L'intera data pipeline[G] ad esclusione della visualizzazione dei dati, è in funzione.

2.1.2 Images

Di seguito sono elencate le immagini Docker[G] utilizzate:

- **Python[G]:**
  - **Image:** python:3.11-bookworm;
  - **Riferimento:** https://hub.docker.com/layers/library/python/3.11-bookworm/images/sha256-94c2dca43c9c127e42dfd021039cc83d8399752097612b49bdc7b00716b6d826 (Ultimo accesso: 2024-08-18)
  - **Ambiente:**
    * Produzione;
    * Test;
    * Stream processing.

- **Apcahe Kafka[G]:**
  - **Image:** bitnami/kafka:3.7.0;
  - **Riferimento:** https://hub.docker.com/r/bitnami/kafka (Ultimo accesso: 2024-08-18)
  - **Ambiente:**
    * Produzione;
    * Test;
    * Stream processing.
- **Apache Flink[G]:**
  - **Image:** flink:1.18.1-java17;
  - **Riferimento:** https://hub.docker.com/_/flink (Ultimo accesso: 2024-08-18)
  - **Ambiente:**
    * Produzione;
    * Development;
    * Stream processing.
- **ClickHouse[G]:**
  - **Image:** clickhouse/clickhouse-server:latest;
  - **Riferimento:** https://hub.docker.com/r/clickhouse/clickhouse-server (Ultimo accesso: 2024-08-18)
  - **Ambiente:**
    * Produzione;
    * Development;
    * Test;
    * Stream processing.
- **Grafana[G]:**
  - **Image:** grafana/grafana-oss:latest;
  - **Riferimento:** https://hub.docker.com/r/grafana/grafana-oss (Ultimo accesso: 2024-08-18)
  - **Ambiente:**
    * Produzione;
    * Development.

2.2 Linguaggi e formato dati

2.2.1 Python[G]

Linguaggio di programmazione ad alto livello, interpretato e multi-paradigma.

- **Versione:** 3.11;
- **Documentazione:** https://docs.python.org/3/ (Ultimo accesso: 2024-08-18)
- **Utilizzo:**
  - Simulazione di dati prodotti da sensori di diverso tipo;
  - Test di unità per singole classi o funzioni;
  - Test d'integrazione tra generatore dati, broker e database utilizzati;
  - Calcolo dei valori per metriche di qualità di prodotto.

- **Librerie e framework[G]:**
  - **pytest:**
    * **Versione:** 8.0.2;
    * **Documentazione:** https://docs.pytest.org/en/7.1.x/contents.html (Ultimo accesso: 2024-08-18)
    * **Utilizzo:** scrittura di codice per test d'integrazione agevolata grazie a funzioni di before_test e after_test e la gestione di fixture.
  - **unittest:**
    * **Versione:** 3.11;
    * **Documentazione:** https://docs.python.org/3/library/unittest.html (Ultimo accesso: 2024-08-18)
    * **Utilizzo:** scrittura di script di test d'unità agevolati dall'uso di patch, asserzioni e set up di casistiche di testing.
  - **numpy:**
    * **Versione:** 1.26.4;
    * **Documentazione:** https://numpy.org/devdocs/user/ (Ultimo accesso: 2024-08-18)
    * **Utilizzo:** fornitura di strutture dati e operazioni matematiche e statistiche complesse.
  - **confluent-kafka:**
    * **Versione:** 2.3.0;
    * **Documentazione:** https://developer.confluent.io/get-started/python/ (Ultimo accesso: 2024-08-18)
    * **Utilizzo:** fornitura di strumenti per la produzione e consumo di messaggi da parte di Apache Kafka.
  - **clickhouse-connect:**
    * **Versione:** 0.7.2;
    * **Documentazione:** https://clickhouse.com/docs/en/integrations/python (Ultimo accesso: 2024-08-18)
    * **Utilizzo:** fornitura di strumenti che semplificano l'interazione con il database[G] ClickHouse[G] permettendo query e inserimenti nello stesso.

2.2.2 SQL[G]

Linguaggio standard per la gestione e la manipolazione dei database[G].

- **Utilizzo:**
  - Creazione delle tabelle del database in base al topic, ovvero la tipologia di sensore;
  - Gestione e interrogazione del database ClickHouse[G].

2.2.3 JSON[G]

Formato per lo scambio di dati, basato sui concetti di oggetto (insieme di coppie chiave-valore) e di array (lista di oggetti, stringhe, valori o array).

- **Utilizzo:**
  - Formato dei messaggi trasmessi dal simulatore dei sensori a Kafka[G];
  - Configurazione delle dashboard[G] e dei pannelli in Grafana[G].

2.2.4 Java[G]

Linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica.

- **Versione:** 17;
- **Documentazione:** https://docs.oracle.com/en/java/ (Ultimo accesso: 2024-08-18)

- **Utilizzo:**
  - Implementazione di Flink[G];
  - Gestione ed elaborazione di dati provenienti da topic[G] di Kafka[G];
  - Reindirizzamento di dati elaborati verso un topic[G] di Kafka.

2.2.5 YAML[G]

Formato di serializzazione leggibile dall'uomo utilizzato per rappresentare dati strutturati in modo chiaro e semplice.

- **Utilizzo:**
  - Configurazione file Docker Compose[G];
  - Configurazione Github[G] workflow e actions;
  - Configurazione provisioning Grafana[G] e politiche di alerting;
  - Configurazione di Flink[G].

2.2.6 XML[G]

Metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

- **Utilizzo:**
  - Definizione di timezone nella configurazione di ClickHouse[G];
  - Definizione di dettagli di configurazione nel POM (Porject Object Model) per l'elaborazione degli stream in Java.

2.3 Servizi della pipeline

2.3.1 Apache Kafka[G]

Apache Kafka è una piattaforma open-source di streaming distribuito sviluppata dall'Apache Software Foundation. Progettata per gestire flussi di dati in tempo reale in modo scalabile e affidabile, è ampiamente utilizzata nel data streaming e nell'integrazione dei dati nelle moderne applicazioni.

- **Versione:** 3.7.0;
- **Documentazione:** https://kafka.apache.org/20/documentation.html (Ultimo accesso: 2024-08-18)
- **Utilizzo:** nel nostro prodotto Kafka agisce da broker[G], ricevendo i dati dai produttori, ovvero il generatore di dati in Python[G] e rendendoli disponibili ai consumer. I consumer sono due:
  - ClickHouse:[G] i dati vengono resi disponibili per essere archiviati nel database[G] suddivisi per topic[G];
  - Flink:[G] i dati sono resi disponibili per analisi e elaborazione in tempo reale attraverso stream processing[G].

  I dati provenienti dalle simulazioni dei sensori vengono inviati a Kafka come messaggi in formato JSON[G].

2.3.2 Apache Flink[G]

Apache Flink è un framework[G] per l'elaborazione di flussi e l'elaborazione in batch open-source sviluppato da Apache Software Foundation. Il nucleo di Flink è un motore di flusso di dati in streaming distribuito scritto in Java e Scala.

- **Versione:** 1.18.1;

- **Documentazione:** https://nightlies.apache.org/flink/flink-docs-stable/ (Ultimo accesso: 2024-08-18)
- **Utilizzo:** nel prodotto, Flink agisce da consumer degli stream dati simulati dal generatore Python[G] e ottenuti da Kafka[G], detto producer. Ottenuti gli stream li elabora e inserisce in un nuovo e apposito topic[G] di Kafka.

2.3.3 ClickHouse[G]

Clickhouse è un sistema[G] di gestione di database[G] (DBMS) di tipo colonnare, progettato per l'analisi di grandi volumi di dati grazie a query SQL[G] in tempo reale.

- **Versione:** 24.2.1.2248;
- **Documentazione:** https://clickhouse.com/docs/en/intro (Ultimo accesso: 2024-08-18)
- **Utilizzo:** nel progetto, Clickhouse ha in carico la persistenza dei dati provenienti dai sensori, recuperandoli in tempo reale da Kafka[G] e assicurandone l'immediata disponibilità per l'analisi. Clickhouse è in grado di gestire grandi volumi di dati in modo rapido e incrementale, grazie all'architettura colonnare. Grafana[G], infine, raccoglie tali dati per la rappresentazione grafica, tramite query SQL.

2.3.4 Grafana[G]

Grafana è una piattaforma open-source per la visualizzazione e l'analisi interattiva di dati, utilizzata per creare dashboard[G] e grafici da fonti di dati eterogenee.

- **Versione:** 10.4.0;
- **Documentazione:** https://grafana.com/docs/grafana/latest (Ultimo accesso: 2024-08-18)
- **Utilizzo:** nel progetto, Grafana viene utilizzato per la visualizzazione dei dati tramite dashboard e grafici interattivi consentendo il monitoraggio in tempo reale dei sensori. Permette inoltre l'analisi di tali dati, fornendo strumenti statistici e opzioni di filtraggio. Infine, offre la possibilità di introdurre un sistema di alerting basato sul verificarsi di condizioni anomale per un tempo prolungato, inviando una notifica sulla piattaforma Discord[G].

3 Architettura di sistema

3.1 Modello architetturale

L'applicativo basa il funzionamento sulla gestione e sull'elaborazione di stream continui di dati eterogenei in tempo reale ma fornendo, al tempo stesso, la possibilità di immagazzinare tali dati per un'analisi su più larga scala di batch dati. Per sviluppare tale sistema[G] abbiamo adottato la k-architecture[G].

3.1.1 k-architecture[G]

L'architettura Kappa è un modello di elaborazione e processamento per la gestione di stream di dati che offre un'alternativa all'architettura Lambda. La caratteristica principale di questa architettura è permettere lo sviluppo sia di stream processing[G] che di batch processing all'interno di un unico stack tecnologico, mantenendo il sistema[G] in real time.

3.1.2 Vantaggi

- Minore complessità nell'implementazione e nella manutenzione;
- Maggiore coerenza tra analisi batch e real-time.

3.1.3 Svantaggi

- Rallentamento dovuto all'elaborazione degli stream in tempo-reale;
- Meno flessibile dell'architettura Lambda per via di un'unico stack tecnologico sulla quale avviene l'elaborazione dei dati.

3.1.4 Componenti di sistema

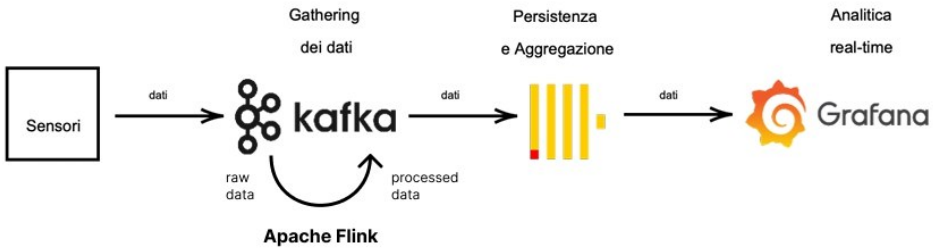

Figura 1: Stack tecnologico dell'applicativo.

- **Data source:** I dati sono generati dal simulatore dei sensori dislocati geograficamente sul territorio della città. La generazione dei dati è event driven[G], quindi basata sul verificarsi di eventi, i quali possono anche corrispondere ad un intervallo di tempo regolare. Il dato generato viene passato sottoforma di messaggio allo streaming layer;
- **Streaming layer:** Lo streaming layer è composto da Apache Kafka[G], un sistema[G] di messaggistica distribuito che riceve i dati dai sensori e li gestisce trattandoli come stream. Successivamente li rende disponibili per l'elaborazione in tempo reale nello streaming layer e per l'archiviazione nello storage layer;
- **Processing layer:** Il processing layer è costituito da Apache Flink[G] che consuma i dati dallo streaming layer e li processa in tempo reale. Successivamente ritorna lo stream allo streaming layer per archiviarlo;
- **Storage layer:** Lo storage layer è costituito dal database colonnare ClickHouse[G], che archivia i dati in arrivo dallo streaming layer e li rende disponibili per l'analisi e la visualizzazione sia in tempo reale che batch nel data visualization layer;

- **Data visualization layer:** Il data visualization layer è composto da Grafana[G], piattaforma di visualizzazione dei dati che si occupa della rappresentazine degli stessi ottenuti dallo storage layer e della gestione delle notifiche in caso di anomalie.

3.2 Data-flow

Il diagramma presentato qui di seguito rappresenta il percorso compiuto da una qualsiasi unità di dati all'interno del sistema, a quali operazioni esso è sottoposto, come viene elaborato e dove viene infine archiviato e utilizzato. Il diagramma mostra dunque anche le connessioni tra le varie parti del sistema e come esse interagiscono tra di loro.

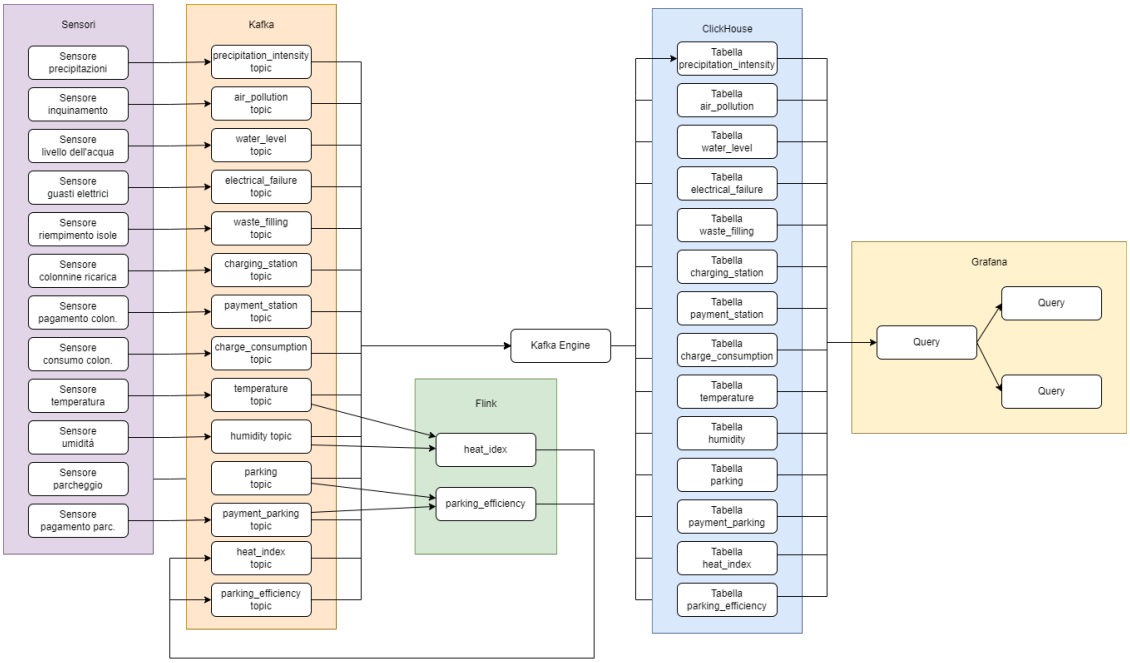

Figura 2: Diagramma del flusso dei dati.

- **Generazione dati:** I dati, prodotti nella realtà dal rilevamento di sensori fisici, vengono qui simulati attraverso dei generatori verosimili nel loro funzionamento e campionamento. I dati generati vengono trasportati poi al broker Kafka[G] in formato JSON[G] e indirizzati ai rispettivi topic[G].
- **Stream processing e instradamento:** Una volta arrivati su Kafka[G] e indirizzati sui topic[G] di appartenenza, i dati vengono direttamente instradati e archiviati su ClickHouse[G] tramite Kafka Engine, materialized view e MergeTree.
  Alcuni dati possono, tuttavia, venire elaborati tramite stream processing[G] attraverso Apache Flink[G], prima di venire reimmessi in un topic[G] apposito e quindi archiviati sul database Clickhouse. Nel nostro caso i dati interessati sono:
  - Temperatura e umidità, sia inviati individualmente che combinati per creare un nuovo tipo di dato, ovvero la temperatura percepita;
  - Pagamenti e Soste di un parcheggio, sia inviati individualmente che combinati per creare un nuovo tipo di dato, ovvero l'efficienza parcheggio.
- **Visualizzazione:** I dati, archiviati su ClickHouse[G], vengono visualizzati da Grafana[G] tramite query sulle tabelle, generando dashboard[G] di visualizzazione per i risultati di tali query.

3.3 Architettura del generatore dati

Il generatore di dati in Pyhton[G] è una componente che punta a simulare l'acquisizione di dati da parte di sensori fisici dislocati nel suolo cittadino. Non è parte effettiva dello stack tecnologico alla quale il capitolato[G] punta ma è parte necessaria per il corretto sviluppo dell'applicativo e per l'analisi del funzionamento delle tecnologie implementate. A seguito, verrà presentata l'architettura individuata mediante l'utilizzo di diagrammi delle classi e relative descrizioni e verranno motivate le scelte dei design pattern individuati in quanto scelte progettuali.

3.3.1 Struttura generale

La struttura generale del simulatore prevede la presenza di tre moduli. Il modulo dei sensori genera i dati su richiesta del modulo dei thread che a sua volta invoca il modulo writer per inviare il dato generato a Kafka[G]. Nella figura è presente solo la classe concreta `TemperatureSensor` che implementa `SensorInterface`, poiché le altre sono analoghe.

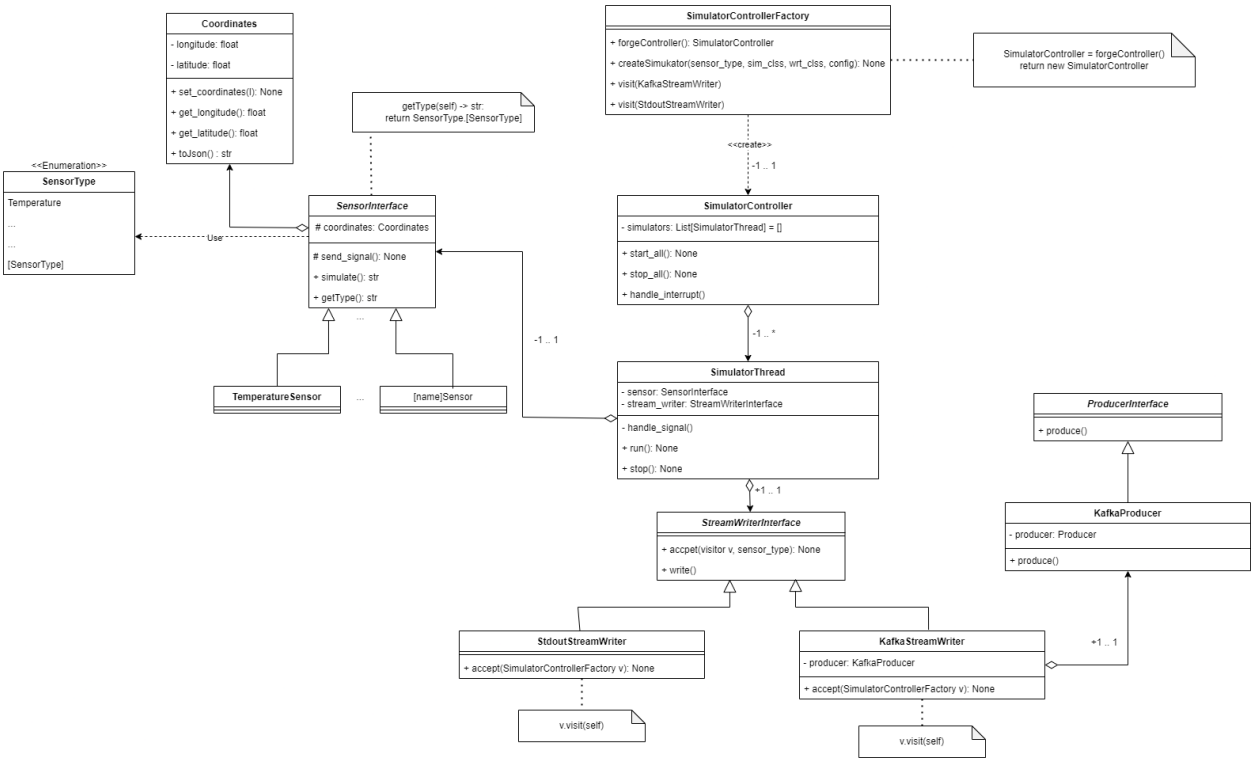

Figura 3: Diagramma delle classi dell'architettura.

3.3.2 Modulo sensori

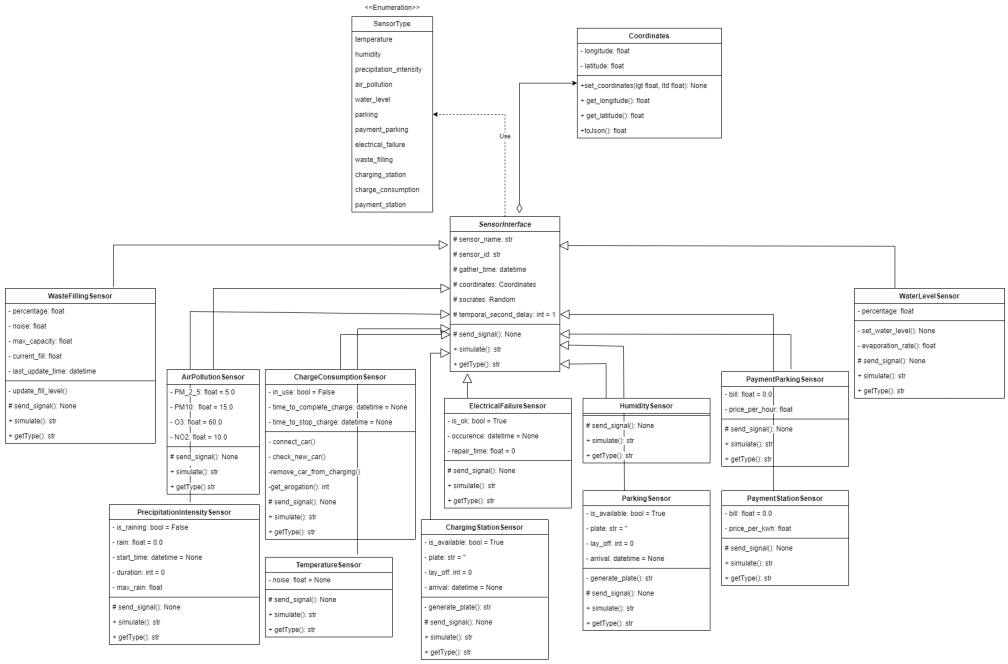

Figura 4: Diagramma delle classi del modulo sensori.

Questo modulo si occupa della generazione dei dati da parte di simulatori verosimili. Lo scopo è quindi quello di diversificare la generazione del dato rispetto ad un interfaccia comune tra tutti i sensori. Design pattern utilizzati:

- **Template method:**
  Questo pattern è implementato nella classe astratta `SensorInterface` la quale offre il comportamento generico di tutti i sensori. Le classi che estendono `SensorInterface` implementano i metodi astratti defininendo un comportamento specifico al tipo di sensore. Questo permette una facile estensione del codice nel caso venga aggiunta una tipologia di sensore e previene la duplicazione di codice. I metodi principali implementati nelle sottoclassi sono due:
  - **simulate():** Genera il dato secondo una logica specifica al tipo di sensore e lo restituisce sottoforma di stringa JSON[G] come valore di ritorno al thread per la sua scrittura;
  - **send_signal():** Metodo protetto che decide la politica di occorrenza di un evento per uno specifico tipo di sensore. Esso può essere un intervallo periodico di tempo, il verificarsi di una condizione casuale o una combinazione dei due. Questo metodo segnala al thread la necessità di effettuare la scrittura di un nuovo dato.

Le classi implementate sono:

- **SensorInterface:**
  - **Attributi:**
    * **sensor_name: string[protected]** - Nome del sensore;
    * **sensor_id: string[protected]** - ID del sensore;
    * **gather_time: datetime[protected]** - Oggetto per metodi datetime;
    * **coorinates: Coorinates[protected]** - Coordinate del sensore;
    * **socrates: Random[protected]** - Oggetto per metodi randomici;
    * **temporal_second_delay: int[protected]** - Delay tra le misurazioni.
  - **Metodi:**
    * **getId(): string[public]** - Funzione che ritorna l'ID del sensore;
    * **simulate(): string[public]** - Metodo astratto per la generazione del dato;

    * **send_signal(): void[protected]** - Metodo per la definizione del meccanismo di evento;
    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.

- **TemperatureSensor:**
  - **Attributi:**
    * **noise: float[private]** - Rumore nei dati.
  - **Metodi:**
    * **simulate(): string[public]** - Metodo che genera il valore di temeperatura in gradi Celsius;
    * **send_signal(): None[protected]** - Metodo per che definisce l'intervallo di tempo tra due simulazioni consecutive;
    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.
- **HumiditySensor:**
  - **Metodi:**
    * **simulate(): string[public]** - Metodo che genera il valore di umidità relativa in percentuale;
    * **send_signal(): None[protected]** - Metodo per che definisce l'intervallo di tempo tra due simulazioni consecutive;
    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.
- **PrecipitationIntensitySensor:**
  - **Attributi:**
    * **is_raining: bool[private]** - Presenza o meno di pioggia;
    * **rain: float[private]** - Valore attuale di intensità;
    * **start_time: datetime[private]** - Momento di inizio della pioggia;
    * **duration: int[private]** - Secondi di durata della pioggia;
    * **max_rain: float[private]** - Massima intensità.
  - **Metodi:**
    * **simulate(): string[public]** - Metodo che genera il valore di intensità in mm/h;
    * **send_signal(): None[protected]** - Metodo per che definisce l'intervallo di tempo tra due simulazioni consecutive;
    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.
- **AirPollutionSensor:**
  - **Attributi:**
    * **PM_2_5: float[private]** - Valore di particolato 2.5;
    * **PM_10: float[private]** - Valore di particolato 10;
    * **O3: float[private]** - Valore di ozono;
    * **NO2: float[private]** - Valore di diossido di azoto.
  - **Metodi:**
    * **simulate(): string[public]** - Metodo che genera il valore di inquinamento dell'aria in µg / m^3;
    * **send_signal(): None[protected]** - Metodo per che definisce l'intervallo di tempo tra due simulazioni consecutive;
    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.
- **WaterLevelSensor:**
  - **Attributi:**
    * **percentage: float[private]** - Valore del livello dell'acqua.
  - **Metodi:**
    * **simulate(): string[public]** - Metodo che genera il valore del livello dell'acqua in percentuale;
    * **send_signal(): None[protected]** - Metodo per che definisce l'intervallo di tempo tra due simulazioni consecutive;

    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.

- **ParkingSensor:**
  - **Attributi:**
    * **is_available: bool[private]** - Disponibilità del parcheggio;
    * **plate: string[private]** - Targa dell'occupante;
    * **lay_off: int[private]** - Tempo di occupazione;
    * **arrival: datetime[private]** - Momento di arrivo.
  - **Metodi:**
    * **generate_plate(): string[private]** - Metodo che genera una targa di un possibile occupante;
    * **simulate(): string[public]** - Metodo che genera la possibilità di occupazione di uno stallo, la targa di chi lo occupa, il periodo di occupazione e il momento in cui è stato occupato;
    * **send_signal(): None[protected]** - Metodo per che definisce l'evento di segnalazione come l'occupazione o la liberazione di uno stallo;
    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.
- **PaymentParkingSensor:**
  - **Attributi:**
    * **bill: float[private]** - Pagamento dovuto;
    * **price_per_hour: float[private]** - Prezzo all'ora.
  - **Metodi:**
    * **simulate(): string[public]** - Metodo che genera il valore del pagamento dovuto in euro;
    * **send_signal(): None[protected]** - Metodo per che definisce l'intervallo di tempo tra segnalazioni consecutive;
    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.
- **ElectricalFailureSensor:**
  - **Attributi:**
    * **is_ok: bool[private]** - Presenza o meno di un gausto;
    * **repair_time: float[private]** - Tempo necessario a riparare un guasto;
    * **occurence: datetime[private]** - Momento di occorenza di un guasto.
  - **Metodi:**
    * **simulate(): string[public]** - Metodo che genera l'occorenza di un guasto, il tempo necessario a ripararlo e il momento in cui è occorso;
    * **send_signal(): None[protected]** - Metodo per che definisce l'evento di segnalazione come l'occorrere e la riparazione di un guasto;
    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.
- **WasteFillingSensor:**
  - **Attributi:**
    * **percentage: float[private]** - Valore di riempimento dell'isola ecologica;
    * **noise: float[private]** - Rumore nei dati;
    * **max_capacity: float[private]** - Capacità massima;
    * **current_fill: float[private]** - Riempimento attuale;
    * **last_update_time: datetime[private]** - Ultimo aggiornamento.
  - **Metodi:**
    * **update_fill_level(): void[private]** - Metodo che aggiorna il valore di riempimento in base all'orario;
    * **simulate(): string[public]** - Metodo che genera il valore di rimepimento dell'isola ecologica in percentuale;
    * **send_signal(): None[protected]** - Metodo per che definisce l'intervallo di tempo tra due segnalazioni consecutive;

    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.

- **ChargingStationSensor:**
  - **Attributi:**
    * **is_available: bool[private]** - Disponibilità della colonnina;
    * **plate: string[private]** - Targa dell'occupante;
    * **lay_off: int[private]** - Tempo di occupazione;
    * **arrival: datetime[private]** - Momento di arrivo.
  - **Metodi:**
    * **generate_plate(): string[private]** - Metodo che genera una targa di un possibile occupante;
    * **simulate(): string[public]** - Metodo che genera la possibilità di occupazione di una colonnina, la targa di chi la occupa, il periodo di occupazione e il momento in cui è stata occupata;
    * **send_signal(): None[protected]** - Metodo per che definisce l'evento di segnalazione come l'occupazione o la liberazione di una colonnina;
    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.
- **PaymentStationSensor:**
  - **Attributi:**
    * **bill: float[private]** - Pagamento dovuto;
    * **price_per_hour: float[private]** - Prezzo all'ora.
  - **Metodi:**
    * **simulate(): string[public]** - Metodo che genera il valore del pagamento dovuto in euro;
    * **send_signal(): None[protected]** - Metodo per che definisce l'intervallo di tempo tra segnalazioni consecutive;
    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.
- **ChargeConsumptionSensor:**
  - **Attributi:**
    * **in_use: bool[private]** - Utilizzo o meno della colonnina;
    * **time_to_complete_charge: datetime[private]** - Tempo alla quale si completa la carica;
    * **time_to_stop_charge: datetime[private]** - Tempo alla quale si ferma la carica;
    * **next_connection: datatime[private]** - Tempo alla quale si collega la prossima macchina;
    * **mean_erogation_power: float[private]** - Erogazione media.
  - **Metodi:**
    * **connect_car(): void[private]** - Metodo che collega la macchina alla colonnina;
    * **remove_car_from_charging(): void[private]** - Metodo che scollega la macchina dalla colonnina;
    * **check_new_car(): void[private]** - Metodo che controlla la disponibilità della colonnina;
    * **get_erogation(): int[private]** - Metodo che ritorna l'erogazione della colonnina;
    * **simulate(): string[public]** - Metodo che genera il valore di consumo di una colonnina di ricarica in kWh;
    * **send_signal(): None[protected]** - Metodo per che definisce l'intervallo di tempo tra due segnalazioni consecutive;
    * **getType(): string[public]** - Funzione che ritorna il tipo di sensore.

3.3.3 Modulo writer

Questo modulo ha il compito di effettuare la scrittura di informazioni su diversi tipi di servizi indipendentemente dal meccanismo di simulazione dei sensori. Design pattern utilizzati:

- **Strategy:**
  Questo pattern è implementato nella classe astratta `StreamWriterInterface` la quale offre diverse strategie per la scrittura dei dati. Sono state implementate due sottoclassi concrete:

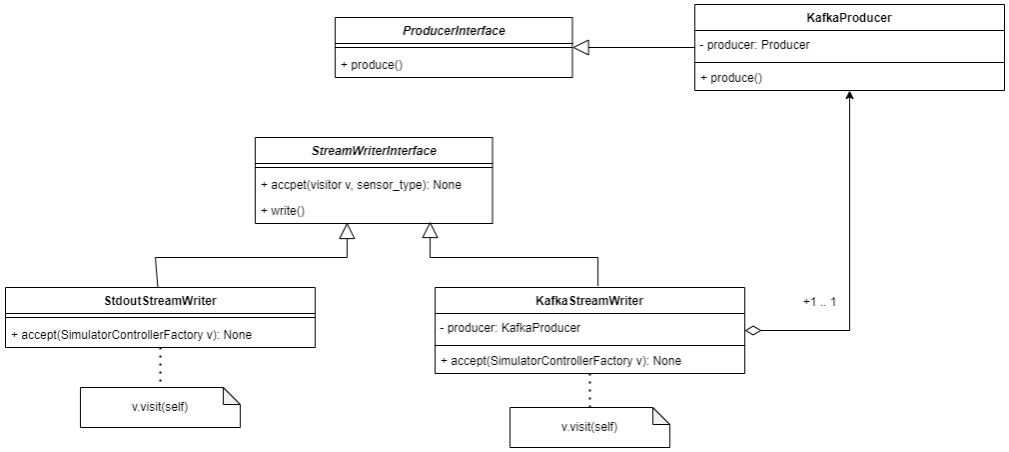

Figura 5: Diagramma delle classi del modulo writer.

  - **StdoutStreamWriter:** Stampa il dato sul terminale;
  - **KafkaStreamWriter:** Invia messaggi ai topic[G] di Kafka[G].

  Quest'ultima classe permette la scrittura chiamando un producer. Con ottica di una possibile estensibilità è previsto che il producer chiamato sia un oggetto della classe `KafkaProducer`, estensione della classe `ProducerInterface`.

Le classi implementate sono:

- **StreamWriterInterface:**
  - **Metodi:**
    * **accept(visitor, sensor_type): void[public]** - Metodo astratto per accettazione del visitor;
    * **write(data: string): void[public]** - Metodo astratto per la scrittura dei dati.
- **StdoutStreamWriter:**
  - **Metodi:**
    * **accept(visitor, sensor, sim_clss, config: Dict): void[public]** - Metodo per accettazione del visitor per standard output;
    * **write(data: string): void[public]** - Metodo per la scrittura dei dati su terminale.
- **KafkaStreamWriter:**
  - **Attributi:**
    * **producer: KafkaProducer[private]** - Producer Kafka;
    * **topic: string[private]** - topic[G] per Kafka.
  - **Metodi:**
    * **accept(visitor, sensor, sim_clss, config: Dict): void[public]** - Metodo per accettazione del visitor per Kafka ourtput;
    * **write(data: string): void[public]** - Metodo per l'invio di dati a topic[G] di Kafka.
- **ProducerInterface:**
  - **Metodi:**
    * **produce(data: string, callback: Callable): void[public]** - Metodo astratto per produrre messaggi per Kafka.
- **KafkaProducer:**
  - **Attributi:**
    * **producer: Producer[private]** - Producer della libreria confluent-kafka;

* **topic: string[private]** - topic[G] per Kafka.

– **Metodi:**

* **produce(data: string, callback: Callable): void[public]** - Metodo per produrre messaggi per Kafka.

3.3.4 Modulo Thread

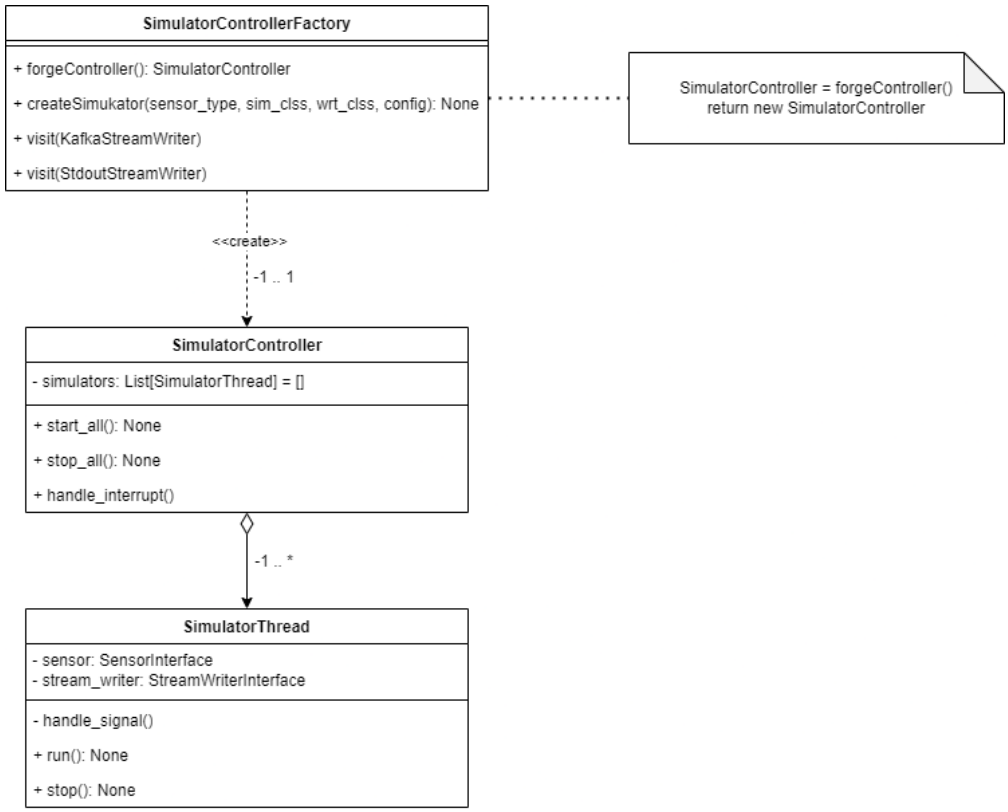

Figura 6: Diagramma delle classi del modulo thread.

Il modulo thread ha lo scopo la gestione dei simulatori come thread asincroni, creati da una factory e gestiti da un controller. Questi funzionano come wrapper dei sensori stessi e hanno il compito di chaimare i writer per la scrittura dei dati generati dai sensori. Design pattern utilizzati:

- **Factory Method:**
  Questo pattern è implementato nella classe `SimulatorControllerFactory` la quale ha il compito di creare tanti thread quanti sensori specificati nel file config.json, senza conoscere il tipo del sensore contenuto. Ha il dovere inoltre di creare il controller per la gestione dei thread;
- **Visitor:**
  Questo pattern è implementato nella classe `SimulatorControllerFactory` la quale permette di riconoscere la classe concreta dello streamwriter per lo specifico thread da creare. Questa pratica può sembrare poco ortodossa ma facilita l'utilizzo del writer per ogni specifico thread.

Le classi implementate sono:

- **SimulatorControllerFactory:**

  – **Attributi:**

  * **simulators: List[SimulatorThread][private]** - Lista di thread creati;
  * **config_file: string[private]** - File di configurazione contenente i sensori;
  * **stream_writer: Type[StreamWriterInterface][private]** - Oggetto streamwriter;

    * **simulators_inventory: Dict[str, int][private]** - Inventario dei sensori creati;
    * **writers_kafka_inventory: Dict[str, KafkaStreamWriter][private]** - Inventario dei tipi di sensori che scrivono su Kafka.
  - **Metodi:**
    * **createSimulator(sensor_type: SensorType, sim_clss: Type[SensorInterface], wrt_clss: Type[StreamWriterInterface], config: Dict): int[protected]** - Metodo che crea un thread;
    * **visit(writer: StdoutStreamWriter, sensor, sim_clss, config: Dict): void[public]** - Metodo per la visita di un sensore con uno standard output writer;
    * **visit(writer: KafkaStreamWriter, sensor, sim_clss, config: Dict, sensor_type: string): void[public]** - Metodo per la visita di un sensore con un kafka writer;
    * **forgeContorller(): SimulatorController[public]** - Metodo per la creazione del controller e dei thread da esso gestiti.

- **SimulatorController:**
  - **Attributi:**
    * **simulators: List[SimulatorThread][private]** - Lista di thread.
  - **Metodi:**
    * **start_all(): void[public]** - Metodo che fa partire tutti i thread;
    * **stop_all(): void[public]** - Metodo che ferma tutti i thread;
    * **handle_interrupt(): void[public]** - Metodo che gestisce le interruzioni e ferma tutti i thread.

- **SimulatorThread**.
  - **Attributi:**
    * **stop_event: Event[private]** - Evento di interruzione;
    * **stream_writer: StreamWriterInterface[private]** - Streamwriter del thread;
    * **sensor: SensorInterface[private]** - Sensore contenuto nel thread.
  - **Metodi:**
    * **handle_signal(): void[private]** - Metodo che scrive il dato generato dal sensore;
    * **run(): void[public]** - Metodo che aspetta il segnale per la simulazione del sensore;
    * **stop(): void[public]** - Metodo che ferma il thead.

3.3.5 Toolkit

Il toolkit è un'insieme di funzioni, classi e costanti che abbaimo utilizzato per agevolare determinate operazioni e/o non strettamente collegati ai moduli precedentemente elencati. Di questo toolkit evidenzieremo:

- **sensor_type:** Enumerazione contenente tutti i tipi di sensori;
- **Coordinates:** Classe che identifica un oggetto coordinata;
  - **Attributi:**
    * **longitude: float[private]** - Longitudine;
    * **latitude: float[private]** - Latitudine.
  - **Metodi:**
    * **setCoordinates(longitude: float, latitude: float): None[public]** - Imposta il valore degli attributi;
    * **get_longitude(): float[public]** - Funzione che ritorna la longitudine;
    * **get_latitude(): float[public]** - Funzione che ritorna la latitudine;
    * **toJson(): string[public]** - Restituisce il contenuto dell'oggetto sottoforma di stringa JSON[G].
- **jsonfy():** Funzione che permette la conversione dei dati di una simulazione in una stringa JSON[G].

3.4 Kafka[G]

3.4.1 Topic

I topic[G] in Kafka[G] permettono di separare logicamente i diversi tipi di messaggi o eventi prodotti. Per separare le diverse misurazioni dei sensori è stato creato un topic[G] dedicato per ogni tipologia di sensore. Ciò facilita la creazione all'interno del database ClickHouse[G] delle tabelle, le quali acquisiscono automaticamente i dati consumandoli da Kafka.

3.4.2 Formato messaggi

I messaggi trasmessi a Kafka[G] sono in formato JSON[G] e possiedono la seguente struttura:

```
{
    "sensor_type": "Tipo sensore",
    "sensor_name": "Nome sensore",
    "sensor_id": "ID sensore",
    "gather_time": "AAAA-MM-DD HH:MM:SS.ssssss",
    "readings": [Oggetti che contengono i valori della misurazione],
    "coordinates": {"type": "point", "coordinates": [Latitudine, Longitudine]}
}
```

3.4.3 Misurazione sensori

Gli oggetti contenenti le misurazioni dei sensori presentano forme diverse in base alla quantità di informazioni passate. Nello specifico, alcuni sensori riferiscono un'unica informazione che viene quindi passata con il seguente formato:

```
{
    "type": "Unità di misura",
    "value": Valore misurato
}
```

I sensori che forniscono più informazioni diverse presentano un messaggio nel formato:

```
{
    "prima informazione": Valore
    "seconda informazione": Valore
}
```

3.5 Flink[G]

Flink costituisce il layer di processamento dei dati. Esso infatti ha il compito di permettere l'elaborazione di stream differenti attraverso i job. Tali job sono stati implementati in java[G].

3.5.1 Job

I job costituiscono una serie di operazioni di trasformazione sui flussi di dati in tempo reale. Tipicamente sono formati da tre componenti:

- **Sorgente:** ingresso del flusso dati;
- **Trasformazioni:** operazioni che modificano i dati in ingresso;
- **Sink:** punto d'uscita del flusso elaborato.

Nel nostro caso le sorgenti sono dei topic[G] Kafka[G]. I flussi ottenuti da questi topic[G] vengono elaborati e poi immessi in un nuovo topic[G] apposito. Per la definizione dei job, Flink mette a disposizine due API: DataStream API, che lavora con gli stream e con maggior controllo sul flusso dati, e Table API, che permette un'approccio basato sulle tabelle con sintassi simile a SQL[G]. Al fine di meglio gestire i flussi di dati, è stato utilizzato DataStream API.

3.5.2 Watermark

I watermark sono un meccanismo utilizzato per tracciare l'avanzamento del tempo degli eventi. Essi sono marcatori di metadati che controllano l'aspetto temporale del flusso durante l'elaborazione. Possiedono un timestamp che specifica quando un evento si è verificato per poi essere confrontato con il tempo in cui tale evento viene elaborato. I watermark sono emessi da Flink nel flusso di dati di origine. In partciolare, un watermark indica che non devono arrivare altri eventi piu vecchi del timestamp del watermark stesso.
Lo scopo è consentire l'elaborazione dei flussi in base al timestamp di produzione dell'evento, importante nel caso in cui gli eventi arrivino fuori ordine. Infine, i watermark agevolano le operazioni basate sul tempo, come l'utilizzo di funzioni di window, che consentono di raggruppare gli eventi in finestre temporali e calcolare risultati basati su tali finestre. Come intervallo di tempo massimo, prima di scartare i dati con watermark precedente al timestamp attuale, è stato definito un arco di 10 secondi.

3.5.3 HeatIndex

Il calcolo della temperatura percepita avviene attraverso l'elaborazione degli stream dati provenienti dai sensori di temperatura e umidità relativa. Va specificato che la presenza di un sensore di temperatura sul suolo cittadino corrisponde alla presenza di un correlato sensore di umidità. Per ogni coppia di sensori è quindi definito un gruppo di appartenenza. Tale scelta è dovuta a necessità empiriche per la comprensione dei fenomeni climatici in luoghi contingenti.
Il job ottiene la temperatura e l'umidità, aggrega i dati dei sensori dello stesso gruppo e successivamente effettua il calcolo dell'heat index, attraverso la formula ideata da Blazejczyk, dove T è temperatura e R è umidità relativa:

$$HI = c_1 + c_2T + c_3R + c_4TR + c_5T_2 + c_6R_2 + c_7T_2R + c_8TR_2 + c_9T_2R_2$$

e i coefficienti c_i sono:

$$c_1 = -8.78469475556\ c_2 = 1.61139411\ c_3 = 2.33854883889$$
$$c_4 = -0.14611605\ c_5 = -0.012308094\ c_6 = -0.0164248277778$$
$$c_7 = 2.211732 * 10^{-3}\ c_8 = 7.2546 * 10^{-4}\ c_9 = -3.582 * 10^{-6}$$

3.5.3.1 Architettura

Per l'implementazione del job è stata creata una classe dotata di un metodo `main` che si occupa di inizializzare l'ambiente di esecuzione e definire sorgente e sink dei dati. Successivamente chiama il metodo `exec` per eseguire il job vero e proprio. Come sorgente di dati, si definiscono i topic[G] relativi alla temperatura e all'umidita, leggendo le variabili d'ambiente relative all'indirizzo del bootstrap server, deserializzate e fornite di watermark. L'esecuzione del job chiama poi le funzioni per l'elaborazione degli stream. Infine pubblica i risultati ad ogni intervallo di tempo definito dalla window, nel topic[G] dedicato allo heat index.
Per ciascuna fase di elaborazione dei dati viene definita una classe che estende la funzione necessaria all'operazione che si vuole effettuare.
Sono stati implementati i seguenti pattern:

- **Template Method:** Questo pattern è presente nelle classi `AbstractTopic` e `ReadingFieldInterface`, le quali forniscono un interfaccia astratta permettendo lo sviluppo di comportamenti diversi nelle sottoclassi attraverso ridefinizione del metodo `toString()`.

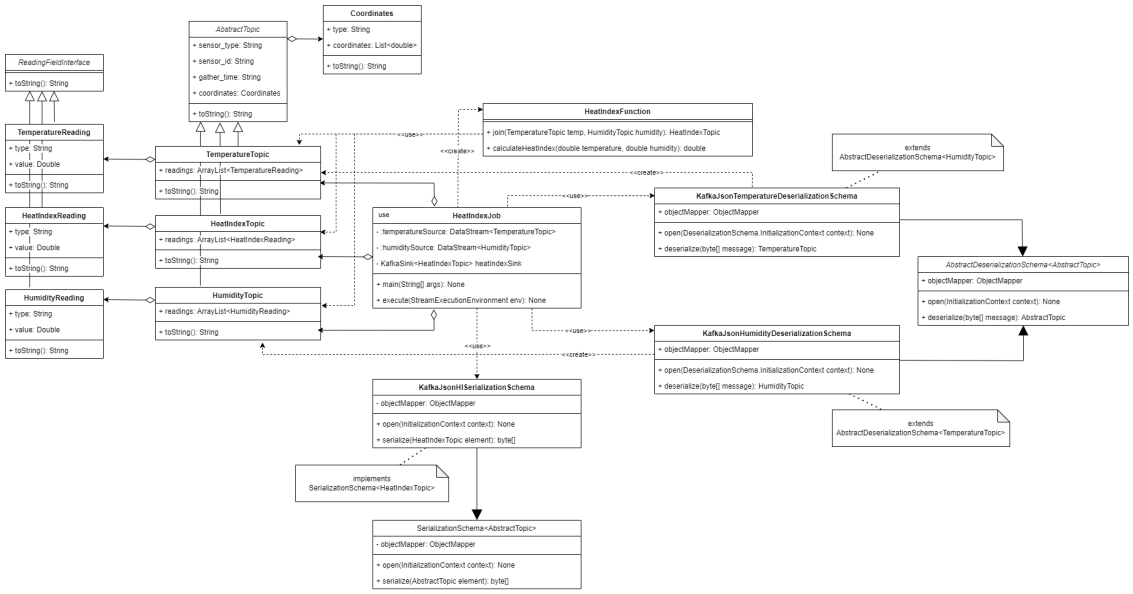

Figura 7: Diagramma delle classi per l'heat index.

3.5.3.2 Classi implementate

- **HeatIndexJob:**
  - **Attributi:**
    - **TEMPERATURE_TOPIC: String[private static final]** - Nome del topic[G] per la temperatura;
    - **HUMIDITY_TOPIC: String[private static final]** - Nome del topic[G] per l'umidità;
    - **HEAT_INDEX_TOPIC: String[private static final]** - Nome del topic[G] per la temperatura percepita;
    - **GROUP_ID: String[private static final]** - Nome del gruppo;
    - **temperatureSource: DataStream¡TemperatureTopic¿[private]** - Sorgente dati per la temperatura;
    - **humiditySource: DataStream¡HumidityTopic¿[private]** - Sorgente dati per l'umidità;
    - **heatIndexSink: KafkaSink¡HeatIndexTopic¿[private]** - Sink Kafka per la temperatura percepita.
  - **Metodi:**

    * **main(args: String[]): void[public,static]** - Metodo principale per inizializzare l'ambiente, definire sources e sink ed eseguire il job;
    * **exec(env: StreamExecutionEnvironment): void[public,static]** - Metodo che applica le trasformazioni.

- **HeatIndexFunction**:
  - **Metodi:**
    * **join(temp: TemperatureTopic, humidity: HumidityTopic): HeatIndexTopic[public]** - Metodo per il join dei dati dei topic[G] di temperatura e umidità per la loro elaborazione e immissione nel topic[G] heat_index;
    * **calculateHeatIndex(temperature: double, humidity: double): double[private]** - Metodo che calcola la temperatura percepita.
- **TemperatureReading:**
  - **Attributi:**
    * **type: String[public]** - Unità di misura;
    * **value: Double[public]** - Valore del dato.
  - **Metodi:**
    * **toString(): String[public]** - Metodo che ritorna una stringa con un dato di temperatura.
- **HumidityReading:**
  - **Attributi:**
    * **type: String[public]** - Unità di misura;
    * **value: Double[public]** - Valore del dato.
  - **Metodi:**
    * **toString(): String[public]** - Metodo che ritorna una stringa con un dato di umidità.
- **HeatIndexReading:**
  - **Attributi:**
    * **type: String[public]** - Unità di misura;
    * **value: Double[public]** - Valore del dato.
  - **Metodi:**
    * **toString(): String[public]** - Metodo che ritorna una stringa con un dato di heat index.
- **TemperatureTopic:**
  - **Attributi:**
    * **readings: ArrayList¡TemperatureReading¿[public]** - Lista di misurazioni di temperatura.
  - **Metodi:**
    * **toString(): String[public]** - Metodo ritorna stringa con un messaggio per topic[G] temperature.
- **HumidityTopic:**
  - **Attributi:**
    * **readings: ArrayList¡HumidityReading¿[public]** - Lista di misurazioni di umidità.
  - **Metodi:**
    * **toString(): String[public]** - Metodo ritorna stringa con un messaggio per topic[G] humidity.
- **HeatIndexTopic:**

  - **Attributi:**
    * **readings: ArrayList¡HeatIndexReading¿[public]** - Lista di misurazioni di temperatura percepita.
  - **Metodi:**
    * **toString(): String[public]** - Metodo ritorna stringa con un messaggio per topic[G] heat_index.

- **KafkaJsonTemperatureDeserializationSchema:**
  - **Attributi:**
    * **serialVersionUID: long[private static final]** - Identificatore di controllo;
    * **objectMapper: ObjectMapper[private transient]** - Oggetto per la serializzazione e deserializzazione per stringhe JSON.
  - **Metodi:**
    * **open(context: InitializationContext): void[public]** - Metodo che crea objectMapper dal contesto;
    * **deserialize(message: byte[]): TemperatureTopic[public]** - Metodo per la deserializzazione del messaggio dal topic[G] temperature.

- **KafkaJsonHumidityDeserializationSchema:**
  - **Attributi:**
    * **serialVersionUID: long[private static final]** - Identificatore di controllo;
    * **objectMapper: ObjectMapper[private transient]** - Oggetto per la serializzazione e deserializzazione per stringhe JSON.
  - **Metodi:**
    * **open(context: InitializationContext): void[public]** - Metodo che crea objectMapper dal contesto;
    * **deserialize(message: byte[]): HumidityTopic[public]** - Metodo per la deserializzazione del messaggio dal topic[G] humidity.

- **KafkaJsonHISerializationSchema:**
  - **Attributi:**
    * **serialVersionUID: long[private static final]** - Identificatore di controllo;
    * **objectMapper: ObjectMapper[private transient]** - Oggetto per la serializzazione e deserializzazione per stringhe JSON.
  - **Metodi:**
    * **open(context: InitializationContext): void[public]** - Metodo che crea objectMapper dal contesto;
    * **serialize(element: HeatIndexTopic): byte[][public]** - Metodo che serializza il messaggio per il topic[G] heat_index e lancia un eccezione se fallisce.

3.5.4 ParkingEfficiency

A partire dai dati riguardanti l'occupazione degli stalli di parcheggio e i pagamenti effettuati su tali stalli, è stato definito un job con il compito di calcolare l'efficienza di tali parcheggi. Anche in questo caso, i sensori riguardanti gli stalli dello stesso parcheggio e il sensore di pagamento per il suddetto fanno parte dello stesso gruppo di sensori.
Il job calcola il rapporto tra il ricavato effettivo e il ricavato desiderabile in un determinato parcheggio. Tale dato, espresso in percentuale, fornisce un'indicazione sull'utilizzo del parcheggio rispetto a quanto previsto alla sua costruzione.
La formula utilizzata è la seguente:

$$Efficiency = \frac{Total\_Revenue}{Total\_Arrivals * Average\_Price}$$

3.5.4.1 Architettura

L'architettura non esula da quanto già specificato nel job per calcolare la temperatura percepita.
Sono stati implementati i seguenti pattern:

- **Template Method:** Questo pattern è presente nelle classi `AbstractTopic` e `ReadingFieldInterface`, le quali forniscono un interfaccia astratta permettendo lo sviluppo di comportamenti diversi nelle sottoclassi attraverso ridefinizione del metodo `toString()`.

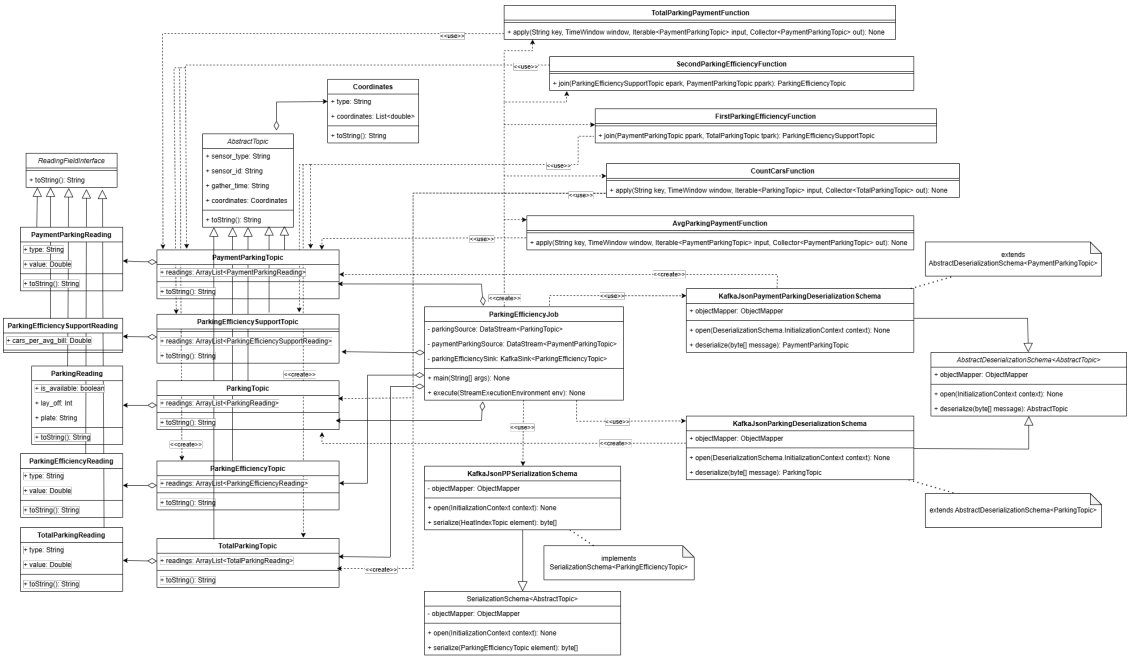

Figura 8: Diagramma delle classi per parking efficiency.

3.5.4.2 Classi implementate

- **ParkingEfficiencyJob:**
  - **Attributi:**
    * **PARKING_TOPIC: String[private static final]** - Nome del topic[G] per i parcheggi;
    * **PAYMENT_PARKING_TOPIC: String[private static final]** - Nome del topic[G] per i pagamenti dei parcheggi;
    * **PARKING_EFFICIENCY_TOPIC: String[private static final]** - Nome del topic[G] per l'efficienza dei parcheggi;
    * **GROUP_ID: String[private static final]** - Nome del gruppo;
    * **parkingSource: DataStream¡ParkingTopic¿[private]** - Sorgente dati per i parcheggi;
    * **paymentParkingSource: DataStream¡PaymentParkingTopic¿[private]** - Sorgente dati per i pagamenti;
    * **heatIndexSink: KafkaSink¡ParkingEfficiencyTopic¿[private]** - Sink Kafka per l'efficienza dei parcheggi.
  - **Metodi:**
    * **main(args: String[]): void[public,static]** - Metodo principale per inizializzare l'ambiente, definire sources e sink ed eseguire il job;
    * **exec(env: StreamExecutionEnvironment): void[public,static]** - Metodo che applica le trasformazioni.

- **AvgParkingPaymentFunction**:
  - **Metodi:**
    * **apply(key: String, window: TimeWindow, input: Iterable¡PaymentParkingTopic¿, out: Collector¡PaymentParkingTopic¿): void[public]** - Metodo per il calcolo del pagamento medio in un parcheggio.
- **CountCarsFunction**:
  - **Metodi:**
    * **apply(key: String, window: TimeWindow, input: Iterable¡PaymentParkingTopic¿, out: Collector¡PaymentParkingTopic¿): void[public]** - Metodo per il calcolo del conteggio delle soste effettuate in un parcheggio.
- **TotalParkingPaymentFunction**:
  - **Metodi:**
    * **apply(key: String, window: TimeWindow, input: Iterable¡PaymentParkingTopic¿, out: Collector¡PaymentParkingTopic¿): void[public]** - Metodo per il calcolo del ricavato totale di un parcheggio.
- **FirstParkingEfficiencyFunction**:
  - **Metodi:**
    * **apply(key: String, window: TimeWindow, input: Iterable¡PaymentParkingTopic¿, out: Collector¡PaymentParkingTopic¿): void[public]** - Metodo per il calcolo dell'efficienza di un parcheggio.
- **SecondParkingEfficiencyFunction**:
  - **Attributi:**
    * **df: DecimalFormat[private static final]** - Formato per la rappresentazione decimale.
  - **Metodi:**
    * **apply(key: String, window: TimeWindow, input: Iterable¡PaymentParkingTopic¿, out: Collector¡PaymentParkingTopic¿): void[public]** - Metodo per l'effettivo calcolo dell'efficienza di un parcheggio.
- **ParkingReading:**
  - **Attributi:**
    * **type: String[public]** - Unità di misura;
    * **value: Double[public]** - Valore del dato.
  - **Metodi:**
    * **toString(): String[public]** - Metodo che ritorna una stringa con un dato riguardante i parcheggi.
- **PaymentParkingReading:**
  - **Attributi:**
    * **type: String[public]** - Unità di misura;
    * **value: Double[public]** - Valore del dato.
  - **Metodi:**
    * **toString(): String[public]** - Metodo che ritorna una stringa con un dato riguardante i pagamenti.
- **TotalParkingReading:**

  - **Attributi:**
    * **type: String[public]** - Unità di misura;
    * **value: Double[public]** - Valore del dato.
  - **Metodi:**
    * **toString(): String[public]** - Metodo che ritorna una stringa con un dato riguardante dei parcheggi.

- **ParkingEfficiencyReading:**
  - **Attributi:**
    * **type: String[public]** - Unità di misura;
    * **value: Double[public]** - Valore del dato.
  - **Metodi:**
    * **toString(): String[public]** - Metodo che ritorna una stringa con un dato riguardante l'efficienza dei parcheggi.

- **ParkingEfficiencySupportReading:**
  - **Attributi:**
    * **cars_per_avg_bill: Double[public]** - Numero di macchine per pagamento medio.

- **ParkingTopic:**
  - **Attributi:**
    * **readings: ArrayList¡ParkingReading¿[public]** - Lista di dati riguardanti i parcheggi.
  - **Metodi:**
    * **toString(): String[public]** - Metodo ritorna stringa con messaggio per topic[G] parking.

- **PaymentParkingTopic:**
  - **Attributi:**
    * **readings: ArrayList¡PaymentParkingReading¿[public]** - Lista di dati riguardanti i pagamenti.
  - **Metodi:**
    * **toString(): String[public]** - Metodo ritorna stringa con messaggio per topic[G] payment_parking.

- **TotalParkingTopic:**
  - **Attributi:**
    * **readings: ArrayList¡TotalParkingReading¿[public]** - Lista di dati riguardanti la totalità dei parcheggi.
  - **Metodi:**
    * **toString(): String[public]** - Metodo ritorna stringa con messaggio per topic[G] total_parking.

- **ParkingEfficiencyTopic:**
  - **Attributi:**
    * **readings: ArrayList¡ParkingEfficiencyReading¿[public]** - Lista di dati riguardanti l'efficienza dei parcheggi.
  - **Metodi:**
    * **toString(): String[public]** - Metodo ritorna stringa con messaggio per topic[G] parking_efficiency.

- **ParkingEfficiencySupportTopic:**
  - **Attributi:**
    * **readings: ArrayList¡ParkingEfficiencySupportReading¿[public]** - Lista di dati a supporto del calcolo dell'efficienza dei parcheggi.
  - **Metodi:**
    * **toString(): String[public]** - Metodo ritorna stringa con messaggio per topic[G] parking_efficiency_support.
- **KafkaJsonParkingDeserializationSchema:**
  - **Attributi:**
    * **serialVersionUID: long[private static final]** - Identificatore di controllo;
    * **objectMapper: ObjectMapper[private transient]** - Oggetto per la serializzazione e deserializzazione per stringhe JSON.
  - **Metodi:**
    * **open(context: InitializationContext): void[public]** - Metodo che crea objectMapper dal contesto;
    * **deserialize(message: byte[]): ParkingTopic[public]** - Metodo per la deserializzazione del messaggio dal topic[G] parking.
- **KafkaJsonPaymentParkingDeserializationSchema:**
  - **Attributi:**
    * **serialVersionUID: long[private static final]** - Identificatore di controllo;
    * **objectMapper: ObjectMapper[private transient]** - Oggetto per la serializzazione e deserializzazione per stringhe JSON.
  - **Metodi:**
    * **open(context: InitializationContext): void[public]** - Metodo che crea objectMapper dal contesto;
    * **deserialize(message: byte[]): PaymentParkingTopic[public]** - Metodo per la deserializzazione del messaggio dal topic[G] payent_parking.
- **KafkaJsonPPSerializationSchema:**
  - **Attributi:**
    * **serialVersionUID: long[private static final]** - Identificatore di controllo;
    * **objectMapper: ObjectMapper[private transient]** - Oggetto per la serializzazione e deserializzazione per stringhe JSON.
  - **Metodi:**
    * **open(context: InitializationContext): void[public]** - Metodo che crea objectMapper dal contesto;
    * **serialize(element: ParkingEfficiencyTopic): byte[][public]** - Metodo che serializza il messaggio per il topic[G] parking_efficiency e lancia un eccezione se fallisce.

3.6 ClickHouse[G]

Il database ClickHouse ha come scopo quello di memorizzare i dati provenienti dai sensori in modo da poterli analizzare e visualizzare in seguito. I dati vengono acquisiti suddivisi per topic, uno per ogni tipo di sensore, e poi memorizzati nelle apposite tabelle.

3.6.1 Struttura

- **Tabella per Kafka topic:** Per ogni tipo di sensore viene creata una tabella contenente una stringa di dati grezzi in formato JSON[G] per ogni messaggio proveniente dal topic[G] dedicato. Questo tipo di tabelle vengono popolate tramite il ClickHouse Kafka Engine, che inserisce all'interno della tabella i messaggi con il topic[G] corrispondente.
  La creazione di tale tabella avviene per provisioning tramite il seguente codice SQL[G] sostituendo il corretto tipo di sensore:

```
CREATE TABLE sc_database.topic_sensor_type
(
    rawJSON String
) ENGINE = Kafka('kafka:9092', 'sensor_type', 'SyncCity', 'JSONAsString');
```

- **Materialized View:** La materialized view, attraverso funzioni di estrazione, estrapola i valori dai dati grezzi delle topic[G] tables e li assegna alle corrispondenti chiavi. Questa view sarà poi fondamentale per la creazione della tabella effettiva del database contenente i dati archiviati. La materialized view è un meccanismo che agevola le query permettendo un accesso semplificato ai dati.
  Vengono create tramite il seguente codice SQL[G], sostituendo il corretto tipo di sensore:

```
CREATE MATERIALIZED VIEW sc_database.sensor_type_mv
TO sc_database.sensor_type
AS
SELECT
    JSONExtractString(rawJSON, 'sensor_name') AS sensor_name,
    JSONExtractString(rawJSON, 'sensor_id') AS sensor_id,
    toDateTime64(JSONExtractString(rawJSON, 'gather_time'), 0) AS gather_time,
    JSONExtractFloat(rawJSON, 'coordinates', 'coordinates', 1) AS latitude,
    JSONExtractFloat(rawJSON, 'coordinates', 'coordinates', 2) AS longitude,
    JSONExtractTipo_di_value(rawJSON, 'readings', 1, 'value') AS value
FROM sc_database.topic_sensor_type;
```

- **Tabella per sensor_type:** Attraverso il motore di archiviazione MergeTree, i dati estrapolati nella materialized view vengono archiviati in una apposita tabella che consiste nel vero e proprio storage di dati, suddiviso per tipo di sensori.
  La creazione delle tabelle avviene attraverso il seguente codice SQL[G] sostituendo il corretto tipo di sensore:

```
CREATE TABLE sc_database.sensor_type
(
    sensor_name String,
    sensor_id String,
    gather_time DATETIME64,
    latitude Float64,
    longitude Float64,
    value Tipo_di_value

) ENGINE = MergeTree()
    ORDER BY (sensor_id, gather_time);
```

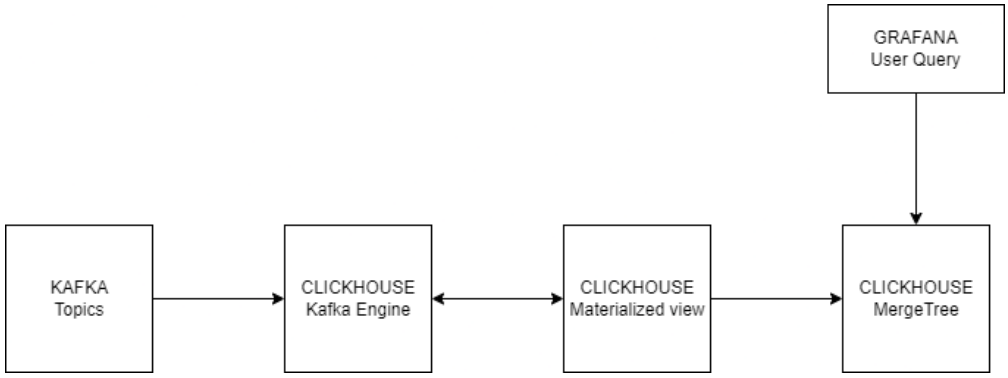

Figura 9: Schema collegamento Kafka ClickHouse.

3.6.2 Motori di archiviazione

Nell'ottica dell'archiviazione dei dati provenienti da Kafka[G], ClickHouse[G] offre vari engine. In particolare, due sono stati utilizzati per la creazione delle tabelle per la persistenza dei dati generati:

- **Kafka Engine:** permette di leggere la stringa JSON[G] contenente i dati grezzi provenienti dal broker e di iniettarla all'interno delle topic[G] tables;
- **MergeTree:** permette di effettuare operazioni di inserimento ed eliminazione in modo efficiente e di effettuare query su intervalli di tempo specifici. Inoltre permette il popolamento delle tabelle con i dati estrapolati nelle materialized view.

3.6.3 Sensore di temperatura

Nella tabella per i sensori di temperatura viene salvato in "readings" il valore di temperatura generato.

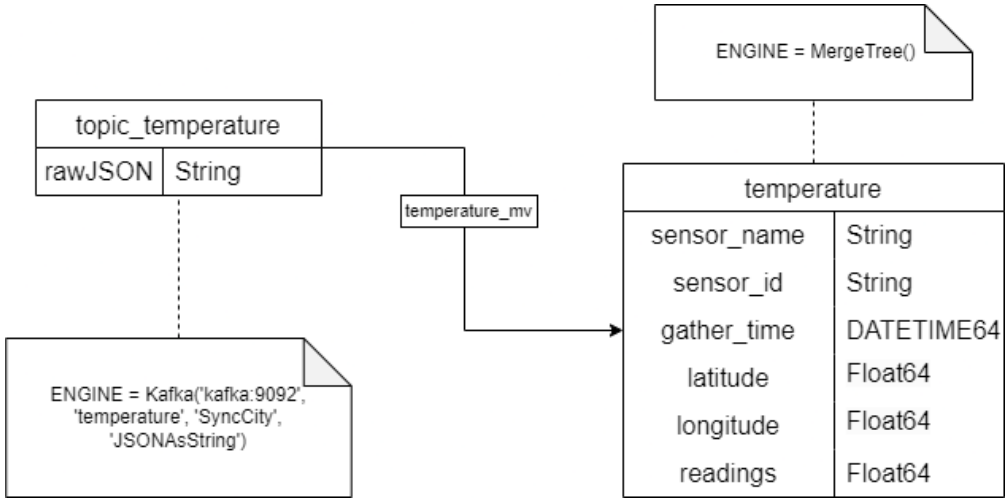

Figura 10: Schema tabelle di tipo temperature.

3.6.4 Sensore di umidità

Nella tabella per i sensori di umidità viene salvato in “readings” il valore di umidità generato.

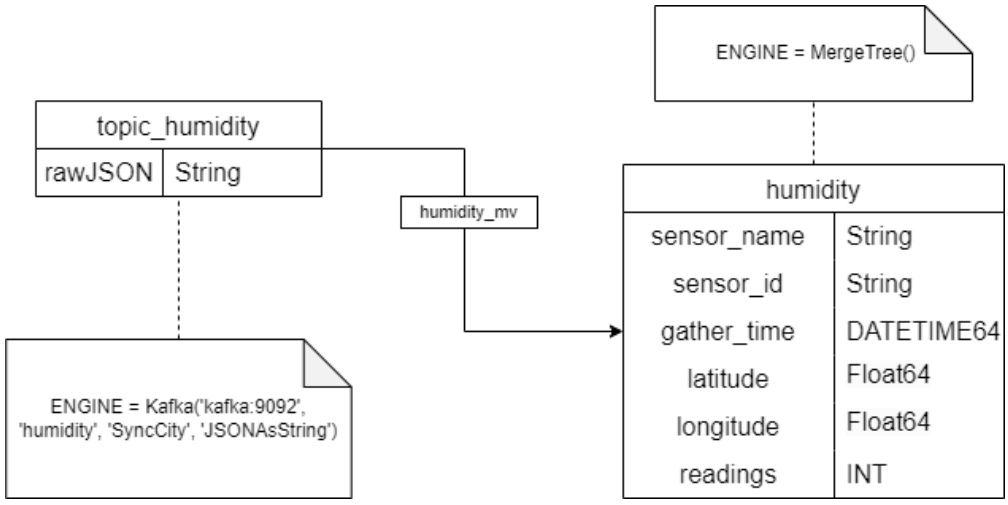

Figura 11: Schema tabelle di tipo humidity.

3.6.5 Sensore di intensità di precipitazione

Nella tabella per i sensori di intensità di precipitazione viene salvato in “readings” il valore di intensità generato.

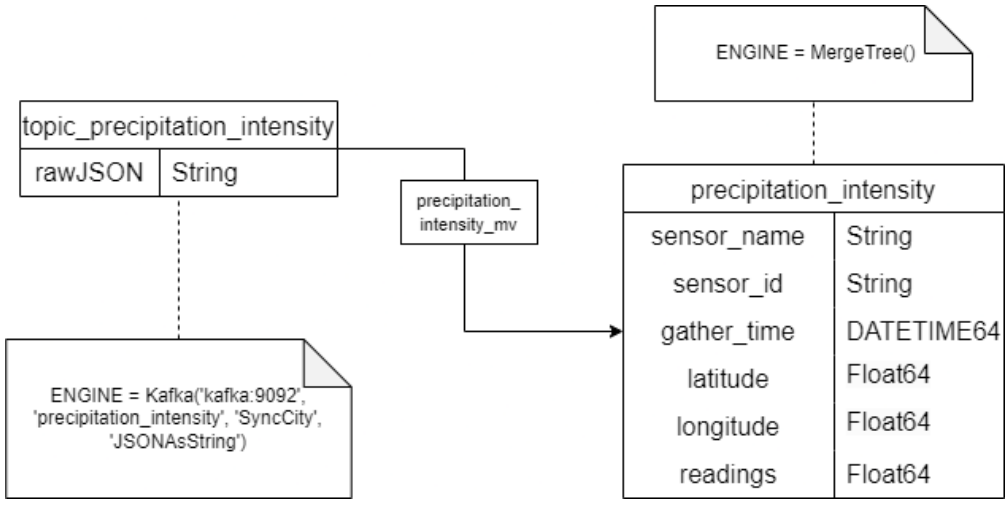

Figura 12: Schema tabelle di tipo precipitation_intensity.

3.6.6 Sensore di inquinamento dell'aria

Nella tabella per i sensori di inquinamento dell'aria vengono salvati i valori di inquinamento dovuto al particolato, all'ozono e al diossido di azoto nelle chiavi "PM2_5", "PM10", "O3" e "NO2".

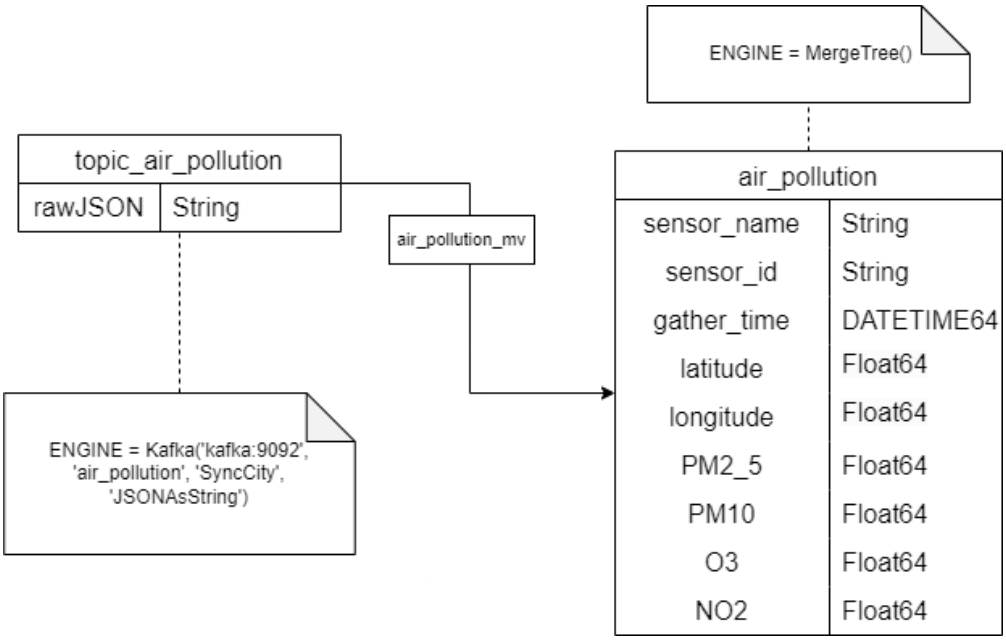

Figura 13: Schema tabelle di tipo air_pollution.

3.6.7 Sensore di misurazione del livello dell'acqua

Nella tabella per i sensori di misurazione del livello dell'acqua viene salvato in "readings" il valore del livello dell'acqua generato.

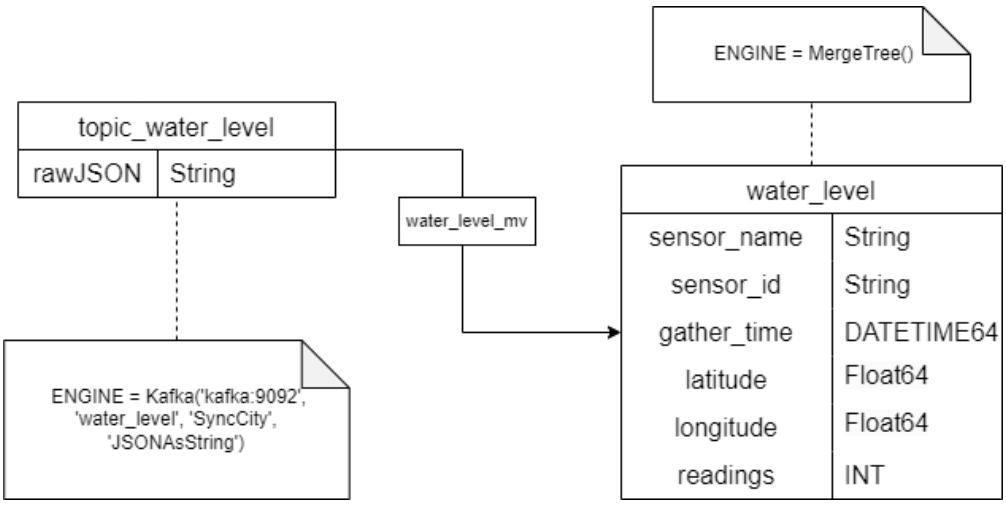

Figura 14: Schema tabelle di tipo water_level.

3.6.8 Sensore di occupazione di parcheggi

Nella tabella per i sensori di occupazione dei parcheggi vengono salvati lo stato di occupazione, la targa dell'occupante e il tempo di permanenza nelle chiavi "is_available", "plate" e "layoff".

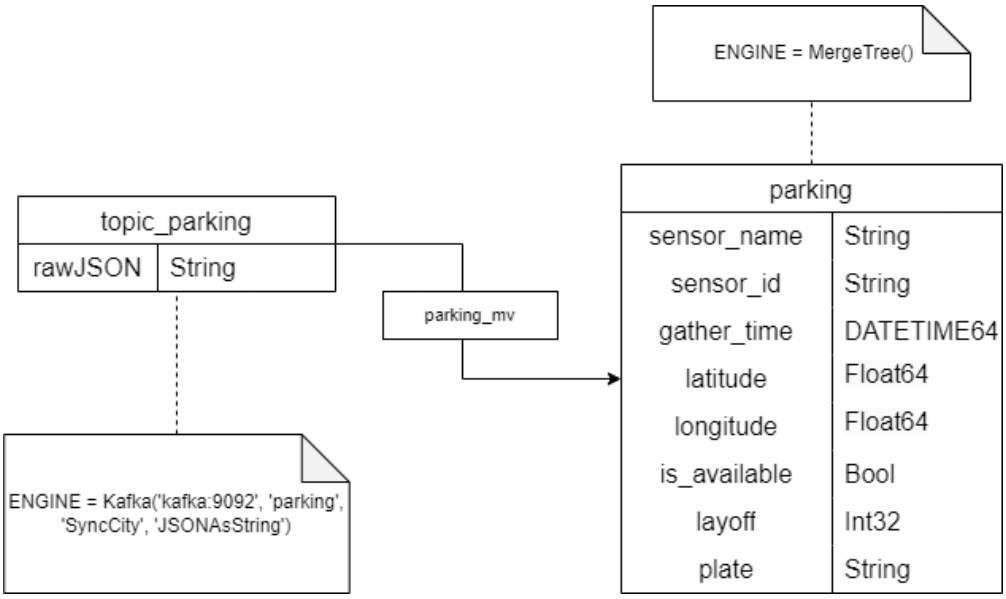

Figura 15: Schema tabelle di tipo parking.

3.6.9 Sensore di pagamento dei parcheggi

Nella tabella per i sensori di pagamento dei parcheggi viene salvato in "bill" l'importo del pagamento generato.

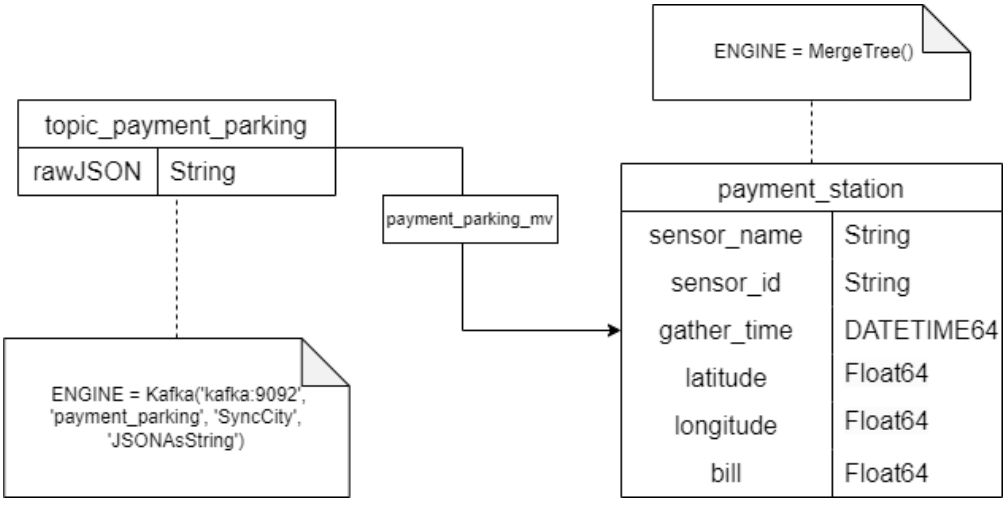

Figura 16: Schema tabelle di tipo payment_parking.

3.6.10 Sensore di guasto elettrico

Nella tabella per i sensori di guasto elettrico vengono salvati lo stato di salute, il momento di occorrenza di un guasto e il tempo di riparazione nelle chiavi “is_ok”, “occurence_fault” e “repair_time”.

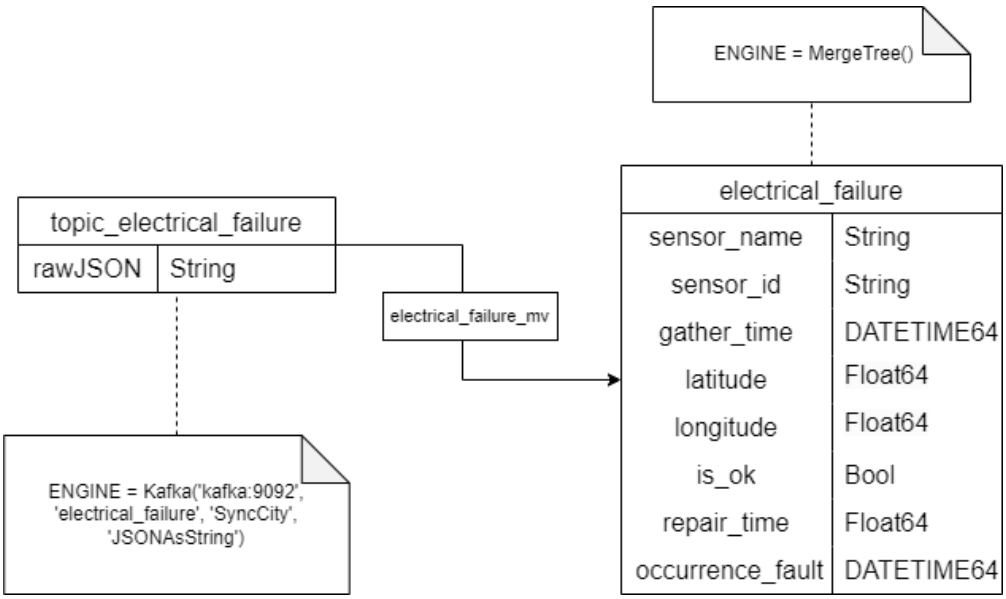

Figura 17: Schema tabelle di tipo electrical_failure.

3.6.11 Sensore di riempimento isole ecologiche

Nella tabella per i sensori di riempimento delle isole ecologiche viene salvato in “readings” il valore di rimepimento generato.

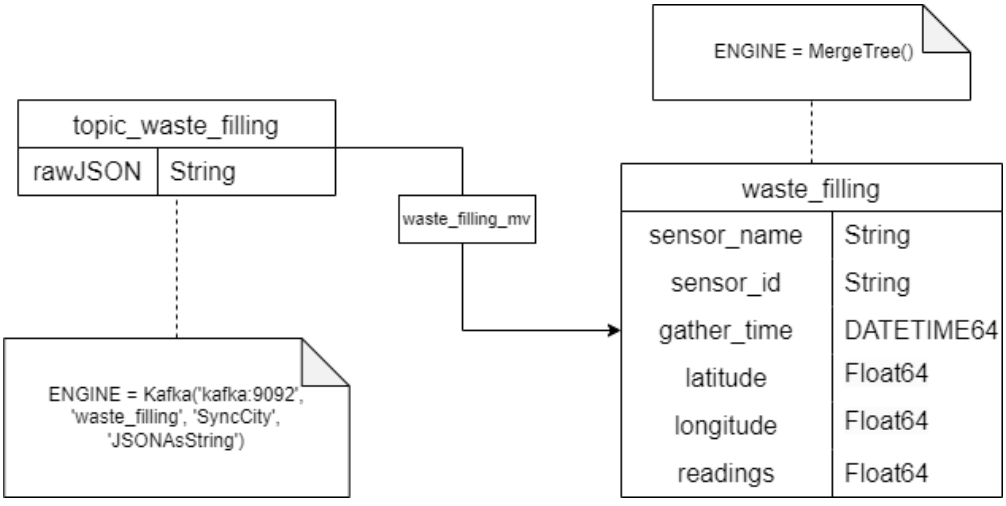

Figura 18: Schema tabelle di tipo waste_filling.

3.6.12 Sensore di occupazione delle colonnine di ricarica

Nella tabella per i sensori di occupazione delle colonnine di ricarica vengono salvati lo stato di occupazione, la targa dell'occupante e il tempo di permanenza nelle chiavi "is_available", "plate" e "layoff".

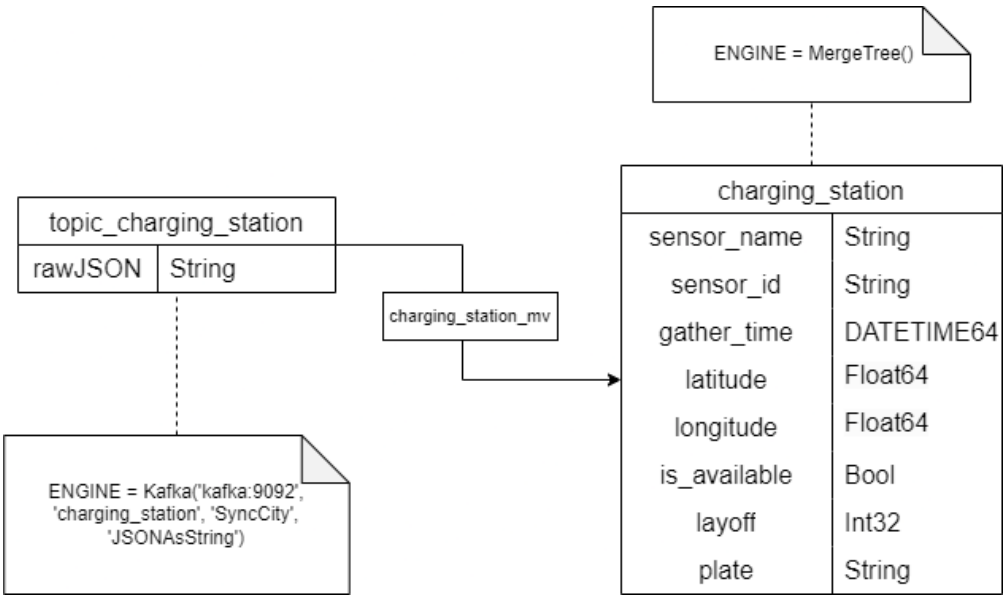

Figura 19: Schema tabelle di tipo charging_station.

3.6.13 Sensore di pagamento delle colonnine di ricarica

Nella tabella per i sensori di pagamento delle colonnine di ricarica viene salvato in "bill" l'importo del pagamento generato.

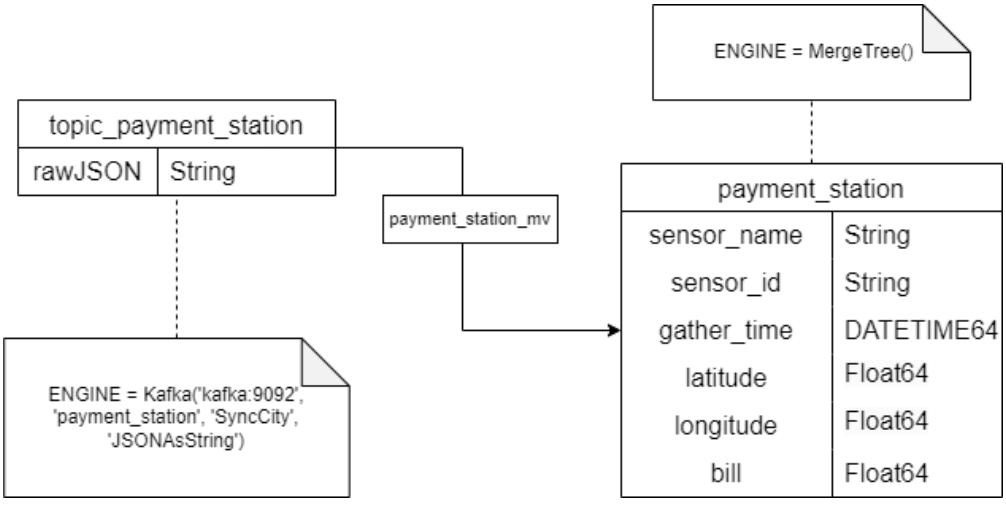

Figura 20: Schema tabelle di tipo payment_station.

3.6.14 Sensore di consumo delle colonnine di ricarica

Nella tabella per i sensori di consumo delle colonnine di ricarica viene salvato in “kwh” il valore di consumo generato.

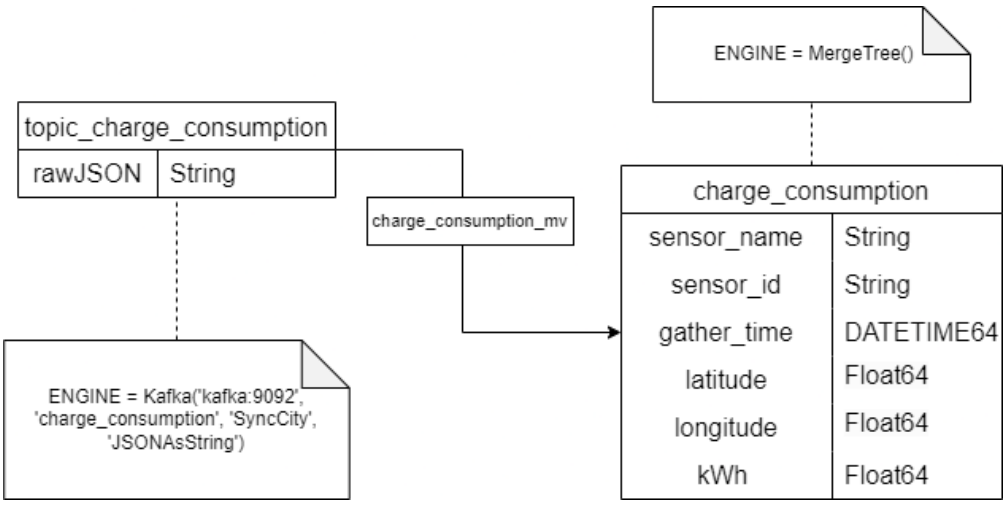

Figura 21: Schema tabelle di tipo charge_consumption.

3.6.15 Calcolo della temperatura percepita

Il calcolo della temperatura percepita avviene attraverso stream processing[G], elaborando gli stream di temperatura e di umidità. Lo stream uscente è poi passato a Kafka[G] come nuovo e dedicato topic[G] e possiede quindi una sua tabella specifica. Il valore di temperatura percepita calcolato è salvato in “readings”.

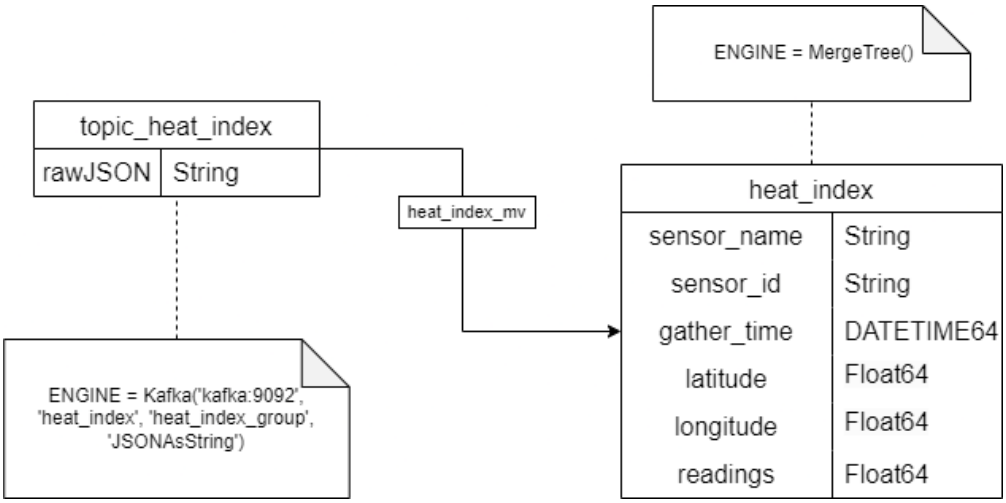

Figura 22: Schema tabelle di tipo heat_index.

3.6.16 Calcolo dell'efficienza dei parcheggi

Il calcolo dell'efficienza dei parcheggi avviene attraverso stream processing[G], elaborando gli stream di occupazione e pagamento dei parcheggi. Lo stream uscente è poi passato a Kafka[G] come nuovo e dedicato topic[G] e possiede quindi una sua tabella specifica. Il valore di efficienza calcolato è salvato in "readings".

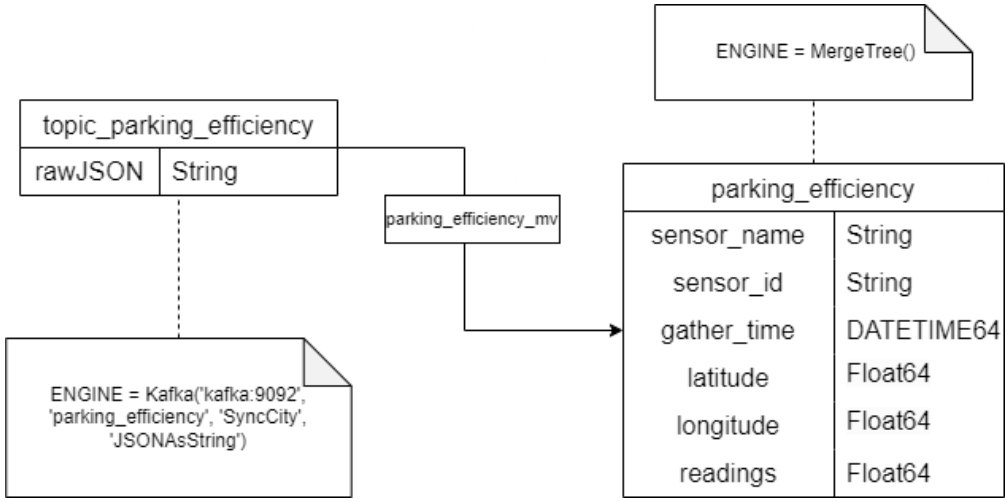

Figura 23: Schema tabelle di tipo parking_efficiency.

3.7 Grafana[G]

Grafana[G] permette la creazione di svariati tipi di dashboard[G] e widget[G] per la rappresentazione grafica dei dati simulati. L'utilizzo di tali dashboard è schermato dalla necessità di autenticarsi attarverso le credenziali fornite dall'amministratore di sistema[G]:

- **Username:** admin;
- **Password:** admin.

3.7.1 Datasource

Come sorgente dati è stato impostato un collegamento con il database ClickHouse[G] grazie ad un plug-in installato attraverso file yaml di configurazione presente in "/provisioning/datasources". Questo plugin di grafana consente di connettersi a un'istanza di ClickHous, di visualizzarne i dati in tempo reale ed eseguire query personalizzate.

3.7.2 Dashboard

I dati rappresentati sono stati categorizzati in base alla tipologia di informazione che forniscono. Sono stati individuati due ambiti:

- **Dati ambientali:** Dati che forniscono informazioni riguardanti il clima e l'ambiente;
- **Dati urbanistici:** Dati che forniscono informazioni riguardanti lo stato dei servizi urbanistici nella città.

Le dashboard[G] implementate sono quattro.

3.7.2.1 Sensors

Dashboard[G] riguardante informazioni generiche sui sensori. Questa dashboard si compone del seguente pannello:

- Pannello geomap[G] per la visualizzazione della locazione geografica dei sensori.

3.7.2.2 Environmental planning

Dashboard[G] riguardante le informazioni di tipo ambientale. Ai fini di una migliore organizzazione dei dati visualizzati e di una più efficace comprensione dei pannelli visualizzati, quest'ultimi sono suddivisi compartimenti attraverso righe comprimibili.

- **Riga Temperature:**
  - Pannello time-series[G] per il monitoraggio dell'andamento della temperatura;
  - Pannello Gauge[G] per il valore medio di temperatura misurata;
  - Pannello a barre per valori statistici, quali media, moda, minimo e massimo di temperatura rilevata.
- **Riga Heat Index:**
  - Pannello time-series[G] per il monitoraggio dell'andamento della temperatura percepita;
  - Pannello Gauge[G] per il valore medio di temperatura percepita;
  - Pannello a barre per valori statistici, quali media, minimo e massimo di temperatura percepita.
- **Riga Humidity:**
  - Pannello time-series[G] per il monitoraggio dell'andamento dell'umidità;
  - Pannello Gauge[G] per il valore medio di umidità misurata.
- **Riga Precipitation Intensity:**

- Pannello time-series[G] per il monitoraggio dell'andamento dell'intensità di precipitazione;
- Pannello Gauge[G] per il valore medio di intensità di precipitazione misurata.

- **Riga Air Pollution:**
  - Pannello time-series[G] per il monitoraggio dell'andamento dei valori di inquinamento dell'aria PM2.5, PM10, O3 e NO2;
  - Pannello time-series[G] per il monitoraggio dell'andamento della quantità di PM2.5 nell'aria;
  - Pannello time-series[G] per il monitoraggio dell'andamento della quantità di PM10 nell'aria;
  - Pannello time-series[G] per il monitoraggio dell'andamento della quantità di O3 nell'aria;
  - Pannello time-series[G] per il monitoraggio dell'andamento della quantità di NO2 nell'aria;
  - Pannello Gauge[G] per il valore medio di PM2.5 misurato;
  - Pannello Gauge[G] per il valore medio di PM10 misurato;
  - Pannello Gauge[G] per il valore medio di O3 misurato;
  - Pannello Gauge[G] per il valore medio di NO2 misurato.

- **Riga Water Level:**
  - Pannello time-series[G] per il monitoraggio dell'andamento del livello dell'acqua;
  - Pannello Gauge[G] per il valore medio del livello dell'acqua misurato.

3.7.2.3 Urban planning

Dashboard[G] riguardante le informazioni di tipo urbanistico. Ai fini di una migliore organizzazione dei dati visualizzati e di una più efficace comprensione dei pannelli visualizzati, quest'ultimi sono suddivisi compartimenti attraverso righe comprimibili.

- **Riga Parking:**
  - Pannello geomap[G] per il monitoraggio della percentuale di occupazione di un parcheggio;
  - Pannello in formato tabellare per tracciare l'occupazione e valutare la disponibilità dei singoli stalli;
  - Pannello a barre per valori statistici, quali media, minimo e massimo pagamento effettuato per l'utilizzo di uno stallo di parcheggio;
  - Pannello in formato di registro per segnalare i pagamaneti effettuati per l'utilizzo degli stalli di parcheggio;
  - Pannello Gauge[G] per il valore di efficienza di un parcheggio.

- **Riga Charging Stations:**
  - Pannello geomap[G] per il monitoraggio della percentuale di occupazione delle colonnine di ricarica;
  - Pannello in formato tabellare per tracciare l'occupazione e valutare la disponibilità delle colonnine di ricarica;
  - Pannello a barre per valori statistici, quali media, minimo e massimo pagamento effettuato per l'utilizzo di una colonnina di ricarica;
  - Pannello in formato di registro per segnalare i pagamaneti effettuati per l'utilizzo delle colonnine di ricarica;
  - Pannello time-series[G] per il monitoraggio dei consumi delle colonnine di ricarica;

- **Riga Waste Fill:**
  - Pannello time-series[G] per il monitoraggio del riempimento delle isole ecologiche.

- **Riga Electrical Failure:**
  - Pannello geomap[G] per il monitoraggio dello stato di salute della linea elettrica.

3.7.2.4 Exceeding Thresholds

Quest'ultima dashboard[G] si concentra sulla visualizzazione degli allarmi attivi. I pannelli sono divisi in due categorie:

- **Riga Environmental Planning:**
  - Pannello alertlist[G] per tracciare lo stato degli allarmi relativi alla temperatura percepita;
  - Pannello alertlist[G] per tracciare lo stato degli allarmi relativi all'intensità di precipitazione;
  - Pannello alertlist[G] per tracciare lo stato degli allarmi relativi all'inquinamento dell'aria da PM10;
  - Pannello alertlist[G] per tracciare lo stato degli allarmi relativi al livello dell'acqua.
- **Riga Urban Planning:**
  - Pannello alertlist[G] per tracciare lo stato degli allarmi relativi al riempimento delle isole ecologiche.

3.7.3 Variables

Le variabili in Grafana[G] rendono le dashboard[G] dinamiche e interattive. Grazie a valori scelti dall'utente permettono di visualizzare dati provenienti da sensori diversi operando da filtro ma anche da metodo di confronto tra sensori dello stesso tipo.
Le variabili nel progetto sono state implementate attraverso i file di provisioning. Nello specifico, nei file JSON[G] che descrivono le dashboard è possibile definire i valori che le variabili nei widget[G] possono assumere.
Le variabili utilizzate riguardano unicamente lo specifico sensore analizzato per tipologia: un tipo di sensore può quindi confrontare nello stesso pannello tutti i sensori della stessa tipologia o concentrarsi nel monitoraggio di uno in particolare. Questo aumenta la versatilità di utilizzo delle dashboard.

3.7.4 Allarmi (Alerts)

Grafana[G] offre un sistema[G] di notifica per avvisare della presenza di anomalie nei dati quando si verificano determinate condizioni. Le notifiche possono essere inviate tramite diversi canali, tra cui email, Telegram[G] e Discord[G].
La configurazione di un allarme avviene impostando una regola tramite query. Esso si attiva quando la condizione impostata viene soddisfatta. Gli allarmi sono configurati per i seguenti eventi:

- Quando un sensore di temperatura segnala una temperatura superiore ai 40°C inferiore ai -10°C;
- Quando un sensore di intensità di precipitazione segnala un'intensità superiore ai 30 mm/h;
- Quando un sensore di inquinamento dell'aria segnala un valore di inquinamento da PM10 superiore a 80 µg / m^3;
- Quando un sensore di rilevazione del livello dell'acqua segnala un livello superiore a 80% della capacità del bacino;
- Quando un sensore di riempimento delle isole ecologiche segnala un riempimento superiore a 80% della capacità del conferitore.

Gli allarmi possono trovarsi i quattro stati:

- **No data:** Non ci sono dati in arrivo dal database;
- **Error:** Ci sono errori nei dati o nelle query dei pannelli;
- **Normal:** Indica che un allarme è disattivato e la sua condizione non è soddisfatta;
- **Pending:**Indica che la condizione per l'attivazione dell'allarme è stata soddisfatta, ma non è ancora trascorso il periodo di valutazione;
- **Firing:** Indica che un allarme è stato attivato, la sua condizione è stata soddisfatta per il periodo di valutazione definito. Viene quindi inviata una notifica ai canali impostati.

Le regole sono configurabili tramite l'interfaccia grafica successivamente inserite in formato yaml in "/provisioning/alerting".

3.7.5 Canali di notifica

La configurazione dei canali di notifica avviene da interfaccia grafica nella sezione “Alerting/Notification channels”. È stato scelto Discord[G] come unico canale di notifica. Per configurare Discord come canale di notifica è necessario:

- Selezionare Discord;
- Configurare il server Discord “SyncCity”;
- Ottenere il webhook URL del canale in: “Impostazioni server/Integrazioni” e selezionare “Visualizza webhook”;
- Inserire il webhook URL del canale;
- Personalizzare il messaggio di notifica.

Le impostazioni di configurazione del canale di notifica sono esportabili in formato yaml e inseribili in “/provisioning/alerting”.

4 Architettura di deployment

L'intero stack tecnologico è stato implementato e configurato attraverso un ambiente Docker[G] che ha permesso lo sviluppo del modello a layer precedentemente descritto e dei suoi servizi. I container[G] che verranno descritti sono presenti nel file *docker-compose.yaml* nella repository[G] del team.

4.1 Data source

La sorgente dati presenta due container:

- **Container pymocksensors:**
  - Esegue il generatore dati;
  - Utilizzato per la produzione dei messaggi JSON[G] indirizzati allo streaming layer.
- **Container pymocksensors-test:**
  - Esegue i test d'unità e d'integrazione.

4.2 Streaming layer

- **Container kafka:**
  - Esegue Kafka[G] per la gestione del flusso dei dati;
  - Utilizzato in produzione dati, per i test d'integrazione e per il testing dello stream processing[G];
  - Accessibile agli altri container attraverso l'indirizo *kafka:9092*.

4.3 Processing layer

- **Container jobmanager:**
  - Gestisce l'esecuzione dei job;
  - Centro nodale di coordinamento di Flink[G].
- **Container taskmanager:**
  - Gestisce le operazioni di stream processing[G] definite.
- **Container flink-deployer:**
  - Esegue Flink e in particolare provvede all'inizializzazione del jobmanager e del taskmanager.

4.4 Storage layer

- **Container clickhouse:**
  - Esegue ClickHouse[G] per l'archiviazione dei dati;
  - Accessibile agli altri container attraverso l'indirizzo *clickhouse:8123*.

4.5 Visualization layer

- **Container grafana:**
  - Esegue Grafana[G] per la visualizzazione dei dati;
  - Accessibile all'esterno nel browser all'indirizzo *localhost:3000*.

5 Tracciamento requisiti

Di seguito è riportato, per ogni requisito, il corrispondente codice, secondo la tabella presente nel documento *Analisi dei Requisiti v2.0.0*, la sua descrizione e se tale requisito è stato soddisfatto.

Codice	Importanza	Descrizione	Stato
RF-1	Obbligatorio	L'utente deve effettuare l'autenticazione per poter usufruire del sistema[G]. Le credenziali di accesso sono fornite dall'amministratore del sistema[G].	Soddisfatto
RF-2	Obbligatorio	Il sistema[G] deve permettere la visualizzazione in tempo reale dei dati provenienti dai sensori.	Soddisfatto
RF-3	Obbligatorio	Il sistema[G] deve integrare molteplici simulatori di sensori in grado di generare dati casuali ma comunque verosimili.	Soddisfatto
RF-4	Obbligatorio	Il sistema[G] deve integrare almeno un sensore[G] che rilevi la temperatura, espressa in gradi Celsius.	Soddisfatto
RF-5	Obbligatorio	Il sistema[G] deve integrare almeno un sensore[G] che rilevi l'umidità nell'aria, espressa in percentuale.	Soddisfatto
RF-6	Obbligatorio	Il sistema[G] deve integrare almeno un sensore[G] che rilevi l'intensità delle precipitazioni, espressa in millimetri orari.	Soddisfatto
RF-7	Obbligatorio	Il sistema[G] deve integrare almeno un sensore[G] che rilevi la quantità di polveri sottili presenti nell'aria, espressa in microgrammi per metro cubo.	Soddisfatto
RF-8	Obbligatorio	Il sistema[G] deve integrare almeno un sensore[G] che rilevi il livello dell'acqua nella zona di installazione, espresso in percentuale.	Soddisfatto
RF-9	Obbligatorio	Il sistema[G] deve integrare almeno un sensore[G] che rilevi lo stato di occupazione dei parcheggi, espresso mediante un valore binario.	Soddisfatto
RF-10	Obbligatorio	Il sistema[G] deve integrare almeno un sensore[G] che invii dati di pagamento per parcheggi.	Soddisfatto
RF-11	Obbligatorio	Il sistema[G] deve integrare almeno un sensore[G] che rilevi la presenza di guasti elettrici, espressa mediante un valore binario.	Soddisfatto

RF-12	Obbligatorio	Il sistema[G] deve integrare almeno un sensore[G] che rilevi lo stato di riempimento delle isole ecologiche, espresso mediante un valore binario.	Soddisfatto
RF-13	Obbligatorio	Il sistema[G] deve integrare almeno un sensore[G] che rilevi lo stato di utilizzo delle colonnine di ricarica, espresso mediante un valore binario.	Soddisfatto
RF-14	Obbligatorio	Il sistema[G] deve integrare almeno un sensore[G] che rilevi il consumo di energia delle colonnine di ricarica.	Soddisfatto
RF-15	Obbligatorio	Il sistema[G] deve integrare almeno un sensore[G] che invii dati di pagamento per colonnine di ricarica.	Soddisfatto
RF-16	Obbligatorio	L'utente deve poter digitare il proprio username nel campo di inserimento corrispondente per accedere al sistema[G].	Soddisfatto
RF-17	Obbligatorio	L'utente deve poter digitare la propria password nel campo di inserimento corrispondente per accedere al sistema[G].	Soddisfatto
RF-18	Obbligatorio	L'utente deve poter visualizzare un messaggio di errore qualora le credenziali di accesso inserite fossero errate.	Soddisfatto
RF-19	Obbligatorio	L'utente deve poter visualizzare un menu attraverso il quale selezionare la dashboard[G] desiderata tra Sensori, Ambientale ed Urbanistica.	Soddisfatto
RF-20	Obbligatorio	L'utente deve poter visualizzare la dashboard[G] relativa allo stato dei sensori.	Soddisfatto
RF-21	Obbligatorio	L'utente deve poter visualizzare un widget[G] contente una mappa su cui è indicata la posizione geografica dei sensori.	Soddisfatto
RF-22	Obbligatorio	L'utente deve poter visualizzare il nome e le coordinate geografiche dei singoli sensori collocati all'interno del territorio urbano.	Soddisfatto
RF-23	Obbligatorio	L'utente deve poter visualizzare la dashboard[G] relativa al dominio ambientale.	Soddisfatto
RF-24	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra le rilevazioni della temperatura in formato time series[G].	Soddisfatto

RF-25	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra le rilevazioni dell'umidità in formato time series[G].	Soddisfatto
RF-26	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra le rilevazioni della temperatura percepita in formato time series[G].	Soddisfatto
RF-27	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra la temperatura media in formato diagramma di Gauge[G] considerando le ultime rilevazioni effettuate dai singoli sensori attivi.	Soddisfatto
RF-28	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra l'umidità media in formato diagramma di Gauge[G] considerando le ultime rilevazioni effettuate dai singoli sensori attivi.	Soddisfatto
RF-29	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra la temperatura percepita media in formato diagramma di Gauge[G] considerando le ultime rilevazioni effettuate dalle coppie di sensori temperatura-umidità attivi.	Soddisfatto
RF-30	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra le statistiche di temperatura in formato diagramma a barre considerando ciascun sensore[G] attivo nell'intervallo di tempo.	Soddisfatto
RF-31	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra le statistiche di temperatura percepita in formato diagramma a barre considerando ciascuna coppia di valori di temperatura e umidità per ogni rispettivo sensore[G] attivo nell'intervallo di tempo.	Soddisfatto
RF-32	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra l'intensità delle precipitazioni in formato time series[G].	Soddisfatto
RF-33	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra l'inquinamento dell'aria in formato time series[G].	Soddisfatto
RF-34	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra il livello dell'acqua in formato time series[G].	Soddisfatto

RF-35	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra l'intensità attuale delle precipitazioni in formato diagramma di Gauge[G] considerando le ultime rilevazioni effettuate dai singoli sensori attivi.	Soddisfatto
RF-36	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra l'inquinamento attuale dell'aria in formato diagramma di Gauge[G] considerando le ultime rilevazioni effettuate dai singoli sensori attivi.	Soddisfatto
RF-37	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra il livello attuale dell'acqua in formato diagramma di Gauge[G] considerando le ultime rilevazioni effettuate dai singoli sensori attivi.	Soddisfatto
RF-38	Obbligatorio	L'utente deve poter visualizzare la dashboard[G] relativa al dominio urbanistico.	Soddisfatto
RF-39	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra una mappa che descrive lo stato di occupazione dei parcheggi.	Soddisfatto
RF-40	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra una tabella che descrive i parcheggi.	Soddisfatto
RF-41	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra una lista di notifiche dei pagamenti dei parcheggi.	Soddisfatto
RF-42	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra una tabella che descrive i guasti alla fornitura elettrica.	Soddisfatto
RF-43	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra lo stato di riempimento delle isole ecologiche in formato time series[G].	Soddisfatto
RF-44	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra una mappa che descrive la disponibilità colonnine di ricarica.	Soddisfatto
RF-45	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra una tabella che le colonnine di ricarica.	Soddisfatto
RF-46	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra i consumi di elettricità delle colonnine di ricarica in formato time series[G].	Soddisfatto

RF-47	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra le notifiche di pagamento relative all'utilizzo delle colonnine di ricarica.	Soddisfatto
RF-48	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra le statistiche relative ai pagamenti dovuti a parcheggi.	Soddisfatto
RF-49	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra le statistiche relative ai pagamenti dovuti a colonnine di ricarica.	Soddisfatto
RF-50	Obbligatorio	L'utente deve poter visualizzare un widget[G] che mostra l'efficienza monetaria dei parcheggi.	Soddisfatto
RF-51	Desiderabile	L'utente deve poter visualizzare la dashboard[G] relativa al superamento delle soglie.	Soddisfatto
RF-52	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra che mostra una lista di notifiche riguardanti il superamento delle soglie di temperatura.	Soddisfatto
RF-53	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra che mostra una lista di notifiche riguardanti il superamento della soglia di precipitazioni.	Soddisfatto
RF-54	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra che mostra una lista di notifiche riguardanti il superamento della soglia di inquinamento dell'aria.	Soddisfatto
RF-55	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra che mostra una lista di notifiche riguardanti il superamento della soglia di livello dell'acqua.	Soddisfatto
RF-56	Desiderabile	L'utente deve poter visualizzare un widget[G] che mostra che mostra una lista di notifiche riguardanti il superamento della soglia di riempimento dei conferitori nelle isole ecologiche.	Soddisfatto
RF-57	Obbligatorio	L'utente deve poter visualizzare un messaggio di errore qualora non sia possibile reperire i dati o vi sia un malfunzionamento nel sistema[G].	Soddisfatto
RF-58	Obbligatorio	L'utente deve poter applicare dei filtri per escludere alcuni dati dalla visualizzazione, secondo determinati criteri.	Soddisfatto

RF-59	Obbligatorio	L'utente deve poter applicare dei filtri per visualizzare solo i dati provenienti da alcune specifiche tipologie di sensori.	Soddisfatto
RF-60	Obbligatorio	L'utente deve poter applicare dei filtri per visualizzare solo i dati misurati in un determinato intervallo temporale.	Soddisfatto
RF-61	Obbligatorio	L'utente deve poter rimuovere i filtri applicati in precedenza in modo da non escludere alcun dato dalla visualizzazione.	Soddisfatto
RF-62	Desiderabile	L'utente deve poter modificare il layout dei widget[G] a proprio piacimento.	Soddisfatto
RF-63	Desiderabile	L'utente deve poter ridimensionare i widget[G] a proprio piacimento.	Soddisfatto
RF-64	Desiderabile	L'utente deve poter spostare i widget[G] a proprio piacimento.	Soddisfatto
RF-65	Obbligatorio	Il sensore[G] deve poter inviare i dati misurati, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto
RF-66	Obbligatorio	Il sensore[G] deve poter inviare i dati relativi alla temperatura, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto
RF-67	Obbligatorio	Il sensore[G] deve poter inviare i dati relativi all'umidità, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto
RF-68	Obbligatorio	Il sensore[G] deve poter inviare i dati relativi alle precipitazioni, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto
RF-69	Obbligatorio	Il sensore[G] deve poter inviare i dati relativi all'inquinamento atmosferico, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto
RF-70	Obbligatorio	Il sensore[G] deve poter inviare i dati relativi al livello dell'acqua, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto
RF-71	Obbligatorio	Il sensore[G] deve poter inviare i dati relativi allo stato di occupazione dei parcheggi, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto

RF-72	Obbligatorio	Il sensore[G] deve poter inviare i dati relativi ai pagamenti dei parcheggi, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto
RF-73	Obbligatorio	Il sensore[G] deve poter inviare i dati relativi ai guasti elettrici, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto
RF-74	Obbligatorio	Il sensore[G] deve poter inviare i dati relativi al livello di riempimento dei conferitori nelle isole ecologiche, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto
RF-75	Obbligatorio	Il sensore[G] deve poter inviare i dati relativi allo stato di occupazione delle colonnine di ricarica, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto
RF-76	Obbligatorio	Il sensore[G] deve poter inviare i dati relativi ai consumi di elettricità delle colonnine di ricarica, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto
RF-77	Obbligatorio	Il sensore[G] deve poter inviare i dati relativi ai pagamenti delle ricariche effettuate mediante colonnine elettriche, assieme al timestamp di rilevazione e alle proprie coordinate geografiche.	Soddisfatto
RF-78	Obbligatorio	Il sistema[G] consentire la correlazione di dati provenienti da sensori diversi.	Soddisfatto
RF-79	Opzionale	Il sistema[G] consentire la previsione di eventi futuri attraverso l'analisi dei dati provenienti dai sensori.	Non soddisfatto

Tabella 1: Requisiti funzionali.

Codice	Importanza	Descrizione	Stato
RQ-1	Obbligatorio	Devono essere rispettati i vincoli e le metriche descritti nel documento *Piano di Qualifica v1.0.0*.	Soddisfatto
RQ-2	Obbligatorio	Devono essere rispettate le norme specificate nel documento *Norme di Progetto v1.0.0*.	Soddisfatto
RQ-3	Obbligatorio	La copertura di test deve essere almeno dell'80% e deve essere adeguatamente documentata.	Soddisfatto
RQ-4	Obbligatorio	La Proponente[G] deve essere in grado di accedere alla repository[G] GitHub[G] del prodotto "SyncCity".	Soddisfatto
RQ-5	Obbligatorio	Il corretto funzionamento dei simulatori di sensori deve essere adeguatamente documentato. Esso deve essere descritto in un documento aggiuntivo appositamente redatto.	Soddisfatto
RQ-6	Obbligatorio	Le scelte implementative e progettuali devono essere motivate ed adeguatamente documentate. Esse dovranno essere descritte in un documento aggiuntivo appositamente redatto.	Soddisfatto
RQ-7	Obbligatorio	I problemi aperti e le eventuali soluzioni da esplorare devono essere adeguatamente documentati. Essi dovranno essere descritti in un documento aggiuntivo appositamente redatto.	Soddisfatto
RQ-8	Obbligatorio	Deve essere fornito il documento *Manuale Utente* che illustra le funzionalità offerte dal sistema[G] e le modalità di fruizione delle stesse.	Soddisfatto
RQ-9	Obbligatorio	Deve essere fornito il documento *Specifica Tecnica* che illustra le scelte architetturali e progettuali effettuate.	Soddisfatto

Tabella 2: Requisiti qualitativi.

Codice	Importanza	Descrizione	Stato
RV-1	Obbligatorio	La persistenza dei dati deve essere realizzata grazie ad un database[G] OLAP[G] colonnare come ClickHouse[G].	Soddisfatto
RV-2	Obbligatorio	Lo stream di dati deve essere gestito mediante un broker[G] come Apache Kafka[G].	Soddisfatto
RV-3	Obbligatorio	La visualizzazione dei dati deve essere consentita utilizzando una piattaforma di data-visualization come Grafana[G].	Soddisfatto
RV-4	Obbligatorio	Lo stream processing[G] deve essere realizzato mediante la tecnologia Apache Kafka.	Soddisfatto
RV-5	Obbligatorio	Il sistema[G] deve essere installato all'interno di un container[G], utilizzando un servizio di virtualizzazione come Docker[G].	Soddisfatto
RV-6	Obbligatorio	I dati generati dal simulatore devono essere in formato JSON[G].	Soddisfatto
RV-7	Obbligatorio	Il sistema[G] deve eseguire correttamente su dispositivi dotati di processore a 64 bit Quad-Core da 2.4 GHz, 6 GB di RAM, 4 GB di spazio di archiviazione libero su disco e una connessione ad internet stabile.	Soddisfatto
RV-8	Obbligatorio	Il sistema[G] deve essere compatibile con le versioni più recenti dei browser web più comuni. Al momento della stesura del presente documento, sono supportate le seguenti versioni: Google Chrome v124, Mozilla Firefox v125, Microsoft Edge v123 e Safari v17.4.	Soddisfatto
RV-9	Obbligatorio	Il sistema[G] deve essere compatibile con le seguenti versioni dei più diffusi sistemi operativi: Windows 10 o 11 compatibile con WSL 2, MacOS 11.0 Big Sur, Ubuntu 22.94, Debian 12, Fedora 38, RHEL 8.	Soddisfatto

Tabella 3: Requisiti di vincolo.

Codice	Importanza	Descrizione	Stato
RP-1	Obbligatorio	Il sistema[G] deve gestire un carico di dati in entrata non inferiore a 50 dati al secondo.	Soddisfatto
RP-2	Obbligatorio	Il sistema[G] deve avere un tempo di elaborazione massimo di 10 secondi, dal momento in cui il dato viene inserito a quando viene effettivamente visualizzato. Non viene considerata la latenza di rete.	Soddisfatto

Tabella 4: Requisiti prestazionali.

5.1 Grafici requisiti soddisfatti

Di seguito è presentato il grafico dei requisiti soddisfatti. Si può vedere come il 99% dei requisiti, ovvero tutti i requisiti obbligatori e desiderabili, siano stati rispettati. L'unico requisito che non ha raggiunto lo stato di adempimento è quello designato come opzionale (RF-79).

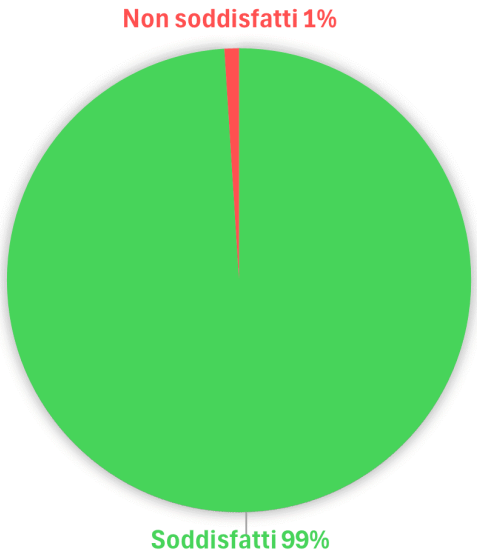

Figura 24: Grafico dei requisiti soddisfatti.