S.W.A.T.
ENGINEERING

Specifica Tecnica

Contatti: swateng.team@gmail.com

Versione: 1.0

Registro delle Modifiche

Versione	Data	Descrizione	Autore	Ruolo	Verificatore
1.0	04-04-2024	Approvazione finale	Riccardo Toniolo	Responsabile	
0.7	21-03-2024	Aggiornamento requisiti soddisfatti e sez. Configurazione visualizzazione e sitema di allerta""	Simone Caregnato	Progettista	Giacomo D'Ovidio, Nancy Kalaj
0.6	19-03-2024	Modifica sezione Database e aggiunta secondo esempio schema, sezione Struttura dei Container, modifica Data-flow diagram	Simone Caregnato	Progettista	Giacomo D'Ovidio, Nancy Kalaj
0.5	19-03-2024	Modifica sezione Tracciamento dei Requisiti	Simone Caregnato	Progettista	Riccardo Toniolo
0.4	18-03-2024	Ridefinizione sezione Architettura dei simulatori	Nancy Kalaj	Progettista	Riccardo Toniolo
0.3	05-03-2024	Stesura sezione Database	Simone Caregnato	Progettista	Riccardo Toniolo
0.2	4-03-2024	Stesura sezioni: Tecnologie, Architettura del sistema	Riccardo Toniolo, Giacomo D'Ovidio, Riccardo Alberto Costantin	Progettista	Simone Caregnato
0.1	21-02-2024	Creazione del documento, impostazione dei grafici, discussione delle tecnologie, creazione parte introduttiva	Riccardo Toniolo, Giacomo D'Ovidio	Progettista	Simone Caregnato

Indice

1 Introduzione 7
1.1 Scopo del documento 7
1.2 Scopo del prodotto 7
1.3 Glossario 7
1.4 Riferimenti 7
1.4.1 Riferimenti normativi 7
1.4.2 Riferimenti informativi 7
1.4.3 Riferimenti tecnici 8
2 Tecnologie 9
2.1 Linguaggi e formati dati 9
2.2 *Framework*$_G$ e librerie 9
2.3 Servizi della pipeline 11
2.4 Analisi e test 12
3 *Architettura*$_G$ del *sistema*$_G$ 13
3.1 Modello architetturale 13
3.2 Diagramma del flusso di dati (*data flow diagram*$_G$) 14
3.3 Struttura dei container 15
3.3.1 Configurazione dei servizi 16
3.3.1.1 Servizio Kafka 16
3.3.1.2 Servizio ClickHouse 16
3.3.1.3 Servizio Grafana 16
3.3.1.4 Servizio Simulators 16
3.3.2 Profili 16
3.4 Database 16
3.4.1 Struttura 17
3.4.1.1 Tabella di accodamento 17
3.4.1.2 Tabella *time series*$_G$ 17
3.4.1.3 Tabella dati aggregati 17
3.4.1.4 Motori di archiviazione 17
3.4.2 Schema 17
3.4.2.1 Sensore per il vento 17
3.4.2.2 Sensore pluviometrico 19
3.5 *Architettura*$_G$ dei simulatori 20
3.5.1 Struttura generale 20
3.5.2 Classi: metodi e attributi 22
3.5.2.1 SimulatorExecutorFactoryTemplate (Classe) 22
3.5.2.1.1 Attributi 22
3.5.2.1.2 Metodi 22
3.5.2.2 KafkaSimulatorExecutorFactory (Classe) 22

3.5.2.2.1 Attributi 22
3.5.2.2.2 Metodi 23
3.5.2.3 StdoutSimulatorExecutorFactory (Classe) 23
3.5.2.3.1 Attributi 23
3.5.2.3.2 Metodi 23
3.5.2.4 SimulatorExecutor (Classe) 23
3.5.2.4.1 Attributi 23
3.5.2.4.2 Metodi 23
3.5.2.5 SimulatorThread (Classe) 23
3.5.2.5.1 Attributi 23
3.5.2.5.2 Metodi 23
3.5.2.6 SensorSimulatorStrategy (Classe) 23
3.5.2.6.1 Attributi 24
3.5.2.6.2 Metodi 24
3.5.2.7 Coordinates (Classe) 24
3.5.2.7.1 Attributi 24
3.5.2.7.2 Metodi 24
3.5.2.8 SensorTypes (Class) 24
3.5.2.8.1 Attributi 24
3.5.2.9 WriterStrategy (interfaccia) 24
3.5.2.9.1 Metodi 24
3.5.2.10 StdOutWriter (Classe) 24
3.5.2.10.1 Metodi 24
3.5.2.11 KafkaWriter (Classe) 24
3.5.2.11.1 Attributi 24
3.5.2.11.2 Metodi 25
3.5.2.12 TargetProducer (Interfaccia) 25
3.5.2.12.1 Metodi 25
3.5.2.13 AdapterProducer (Interfaccia) 25
3.5.2.13.1 Attributi 25
3.5.2.13.2 Metodi 25
3.6 Messaggi ed Eventi 25
3.6.1 Kafka Topics 25
3.6.2 Struttura dei messaggi 25
3.7 Configurazione visualizzazione e sistema di allerta 27
3.7.1 Dashboard 27
3.7.2 Sistema di notifica 27
4 Tracciamento dei requisiti 29
4.1 Tabella dei requisiti soddisfatti 29
4.2 Grafici requisiti soddisfatti 36

Elenco delle Figure

Figure 1: Schema delle componenti del sistema. 13
Figure 2: Diagramma data flow relativo al percorso dei dati. 14
Figure 3: Schema delle tabelle per il *sensore*$_G$ del vento 18
Figure 4: Schema delle tabelle per il *sensore*$_G$ pluviometrico 19
Figure 5: Diagramma delle classi 2 20
Figure 6: Diagramma delle classi 2 21
Figure 7: Diagramma delle classi 3 22

Elenco delle Tabelle

Table 1: Tabella tecnologie per la Codifica: Linguaggi e formati dati. 9
Table 2: Tabella tecnologie per la Codifica: *Framework*$_G$ e librerie 10
Table 3: Tabella tecnologie per la Codifica: Servizi della pipeline. 11
Table 4: Tabella tecnologie per l'analisi del codice. 12
Table 5: Tabella dei requisiti funzionali con annesso stato di soddisfacimento. . 29
Table 6: Tabella dei requisiti di vincolo con annesso stato di soddisfacimento. . 35
Table 7: Tabella dei requisiti di qualità con annesso stato di soddisfacimento. . 35

1 Introduzione

1.1 Scopo del documento

Il presente documento si propone come risorsa esaustiva per la comprensione degli aspetti tecnici chiave del progetto "InnovaCity". La sua finalità primaria è fornire una descrizione dettagliata e approfondita dell'architettura implementativa del *sistema*$_G$. Nel contesto dell'*architettura*$_G$, si prevede un'analisi approfondita che si estenda anche al livello di design più basso. Ciò include la definizione e la spiegazione dettagliata dei *design pattern*$_G$ e degli idiomi utilizzati nel contesto del progetto. L'obiettivo principale del presente documento è triplice: innanzitutto, motivare le scelte di sviluppo adottate; in secondo luogo, fungere da guida fondamentale per l'attività di codifica; infine, monitorare la copertura dei requisiti identificati nel documento *Analisi dei Requisiti v2.0*. Alla luce del modello di sviluppo agile individuato dal team, la redazione del documento segue un approccio iterativo. L'adeguatezza del documento e dell'*architettura*$_G$ individuata viene costantemente monitorata e modificata sulla base dei requisiti e dei feedback ricevuti da parte della Proponente.

1.2 Scopo del prodotto

Lo scopo del prodotto è la realizzazione di un sistema di persistenza dati e successiva visualizzazione di questi, provenienti da sensori dislocati geograficamente. Tale piattaforma consentirà all'*amministratore pubblico*$_G$ di acquisire una panoramica completa delle condizioni della città, facilitando così la presa di decisioni informate e tempestive riguardo alla gestione delle risorse e all'implementazione di servizi.

1.3 Glossario

Al fine di evitare possibili ambiguità relative al linguaggio utilizzato nei documenti, viene fornito il *Glossario v2.0*, nel quale sono presenti tutte le definizioni di termini aventi uno specifico significato che vuole essere disambiguato. Tali termini, sono scritti in corsivo e marcati con una G a pedice.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- Capitolato C6 - InnovaCity: Smart city monitoring platform:

https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C6.pdf (21/03/2024)
https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C6p.pdf (20/02/2024)

- *Norme di Progetto v2.0*
- *Analisi dei Requisiti v2.0*
- Regolamento progetto didattico:

https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/PD2.pdf (20/02/2024)

1.4.2 Riferimenti informativi

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). **Design Patterns: Elements of Reusable Object-Oriented Software.** Addison-Wesley.
- 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 (4/03/2024)

- 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 (20/02/2024)
- Progettazione - corso di Ingegneria del Software a.a. 2023/2024: https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T6.pdf (21/03/2024)
- Architetture κ & λ:
  - https://imply.io/blog/building-event-analytics-pipeline-with-confluent-cloud-and-imply-real-time-database-polaris/ (28/02/2024)
  - https://medium.com/sysco-labs/real-time-sales-analytics-using-kappa-architecture-852dc84bfe7b (28/02/2024)

1.4.3 Riferimenti tecnici

- ***Python***$_G$: https://docs.python.org/3/ (4/03/2024)
- ***Grafana***$_G$: https://grafana.com/docs/grafana/latest/ (21/03/2024)
- **Apache *Kafka***$_G$: https://kafka.apache.org/20/documentation.html (4/03/2024)
- ***Clickhouse***$_G$: https://clickhouse.com/docs (4/03/2024)
- **Confluent Kafka**:https://docs.confluent.io/kafka-clients/python/current/overview.html (20/02/2024)
- **Docker**: https://docs.docker.com/ (4/03/2024)

2 Tecnologie

Questa sezione si propone di offrire una panoramica sintetica delle tecnologie adottate. Questo breve paragrafo introduttivo fornisce un quadro generale delle piattaforme, dei linguaggi di programmazione, dei *framework*$_G$ e di altre risorse tecnologiche che costituiscono le basi del nostro progetto.

2.1 Linguaggi e formati dati

Tecnologia	Descrizione	Uso	Versione/Standard
Python$_G$	Linguaggio di programmazione ad alto livello, interpretato, multi paradigma.	Creazione dei simulatori di dati.	3.11
SQL	Linguaggio standard per la gestione e la manipolazione dei database che lo supportano.	Gestione e manipolazione del database *ClickHouse*$_G$.	ANSI SQL
YAML	Formato di serializzazione dei dati leggibile dall'uomo comunemente utilizzato per la configurazione dei servizi e lo scambio di dati tra programmi.	Configurazione di *Docker Compose*$_G$.	1.2.2
JSON$_G$	Formato leggero per lo scambio di dati, facile da leggere e scrivere per gli esseri umani e facile da analizzare e generare per le macchine.	Configurazione dei simulatori di dati, formato dei messaggi spediti dai simulatori al *broker*$_G$ dati, configurazione delle dashboard *Grafana*$_G$.	2020-12

Table 1: Tabella tecnologie per la Codifica: Linguaggi e formati dati.

2.2 *Framework*$_G$ e librerie

Tecnologia	Descrizione	Uso	Versione
Confluent Kafka	Libreria *Python*$_G$ progettata per agevolare la produzione e il consumo di messaggi all'interno di un ambiente *Apache Kafka*$_G$. Questa libreria fornisce agli sviluppatori gli strumenti necessari per interagire con efficacia con *Kafka*$_G$, consentendo loro di scrivere codice *Python*$_G$ per inviare e ricevere messaggi tramite il sistema di messaggistica distribuita di *Kafka*$_G$. Grazie a Confluent Kafka, è possibile implementare in modo efficiente la comunicazione asincrona e la gestione dei flussi di dati all'interno delle applicazioni *Python*$_G$, sfruttando le potenzialità di *Apache Kafka*$_G$ per garantire scalabilità, affidabilità e prestazioni ottimali.	Interfaccia del codice *Python*$_G$ dei simulatori con il message *broker*$_G$ *Kafka*$_G$.	1.9

Table 2: Tabella tecnologie per la Codifica: *Framework*$_G$ e librerie

2.3 Servizi della pipeline

Tecnologia	Descrizione	Uso	Versione
Clickhouse$_G$	Sistema di gestione dei database (DBMS) colonnari, progettato per l'analisi dei dati in tempo reale. È ottimizzato nei casi d'uso OLAP per eseguire query analitiche su grandi volumi di dati in modo efficiente.	Archiviazione e estrazione di dati aggregati.	24.1.5.6
ClickHouse Kafka Table Engine	Motore di archiviazione di *ClickHouse*$_G$ che consente di leggere i dati da un server *Kafka*$_G$ e di archiviarli in tabelle di *ClickHouse*$_G$.	Svolge il ruolo di consumatore per *Kafka*$_G$ per il recupero e l'archiviazione dei dati.	24.1.5.6
Apache Kafka$_G$	Piattaforma di streaming di dati distribuita e scalabile, progettata per la gestione di flussi di dati in tempo reale. È ampiamente utilizzato per l'elaborazione di eventi, la messaggistica asincrona e la creazione di pipeline dati *real-time*$_G$.	Riceve in modo asincrono i dati provenienti dai simulatori in formato *JSON*$_G$ e li rende disponibili ai suoi consumatori.	3.7.0
Grafana$_G$	Piattaforma open source per il monitoraggio e l'analisi dei dati. Fornisce strumenti per la visualizzazione di metriche e log, la creazione di *dashboard*$_G$ interattive e la generazione di avvisi in tempo reale.	Visualizzazione sottoforma di *dashboard*$_G$ e analisi dei dati recuperati dal database.	10.3
Grafana ClickHouse Data Source	Plugin per *Grafana*$_G$ che consente di interrogare e visualizzare i dati di *ClickHouse*$_G$ in *Grafana*$_G$.	Interrogazione al database di Clickhouse per la visualizzazione dei dati in *Grafana*$_G$.	4.0.3
Docker	Piattaforma open-source che permette di creare, distribuire e gestire applicazioni in contenitori virtuali.	Creare in modo riproducibile ambienti software per i simulatori, *ClickHouse*$_G$, *Kafka*$_G$ e *Grafana*$_G$.	25.0.3
Docker Compose$_G$	Strumento per definire e gestire applicazioni multi-container Docker attraverso file YAML.	Gestione automatica e centralizzata dei container che costituiscono il *sistema*$_G$.	2.24.6

Table 3: Tabella tecnologie per la Codifica: Servizi della pipeline.

2.4 Analisi e test

Tecnologia	Descrizione	Uso	Versione
PEP8	Stile di codifica per il codice *Python*$_G$ che definisce le linee guida per la formattazione del codice, rendendolo più leggibile e uniforme.	Miglioramento stilistico del codice.	1.7.1
Pylint	Strumento di analisi statica del codice *Python*$_G$. Attraverso la scansione del codice, identifica potenziali errori, incongruenze stilistiche e altre possibili problematiche; aiuta a garantire la correttezza, la leggibilità e la manutenibilità del codice.	Analisi statica.	3.1.0
Pytest	Framework di test progettato per *Python*$_G$, noto per la sua flessibilità e semplicità d'uso. Offre agli sviluppatori un ambiente intuitivo per scrivere e eseguire test per verificare la correttezza e l'affidabilità del codice. Con Pytest è possibile effettuare una vasta gamma di test, tra cui test di unità, integrazione e accettazione, garantendo una copertura completa dei casi di test.	Test di unità e test di integrazione	8.1.1

Table 4: Tabella tecnologie per l'analisi del codice.

3 *Architettura*$_G$ del *sistema*$_G$

3.1 Modello architetturale

Il *sistema*$_G$ richiede la capacità di processare dati provenienti da varie fonti, in tempo reale, e di offrire una visualizzazione immediata e continua di tali dati, consentendo di monitorarne gli andamenti. Per questo tipo di scopo le due architetture consigliate sono la:

- **λ-architecture**: che prevede di elaborare i dati in due flussi separati, uno per i dati in tempo reale (streaming) e uno per i dati storici. I dati in tempo reale vengono elaborati immediatamente per fornire risposte rapide, mentre i dati storici vengono elaborati in batch per fornire risposte più complete o elaborate nel tempo. Alla fine, i risultati dei due flussi vengono combinati per fornire una visione completa dei dati;
  - **Vantaggi**: permette di elaborare i dati in tempo reale e batch, è fault tolerant e consistente tra i due flussi;
  - **Svantaggi**: complessità di gestione del *sistema*$_G$, logica di computazione duplicata e complessità di manutenzione;
- **κ-architecture**: semplifica la struttura della λ-architecture, rimuovendo il flusso di batch, lasciando solamente quello di stream. Tutti i dati vengono quindi elaborati in tempo reale utilizzando un sistema di elaborazione dei dati in streaming. Questo fa si che i dati vengono elaborati una sola volta, riducendo la complessità complessiva del *sistema*$_G$;
  - **Vantaggi**: riduzione della complessità del *sistema*$_G$, logica di computazione non duplicata e complessità di manutenzione ridotta, inoltre permette una bassa latenza e scala facilmente con il volume dei dati;
  - **Svantaggi**: non adatto per tutti i casi d'uso, in quanto non permette di elaborare i dati in batch.

Per quanto appena descritto, la κ-architecture è la soluzione più adatta, in quanto specifica per il nostro caso d'uso. I vantaggi dati dalla λ-architecture non risultano utili per i nostri fini; Risulta invece vantaggioso avere un'unica pipeline di elaborazione dei dati in streaming. Si ha appunto bisogno di lavorare con dati in tempo reale, evitando di gestire la lavorazione dei dati in due posti diversi con diverse tecnologie, che porterebbe ad una complessità di manutenzione maggiore, logica di computazione duplicata e complessità di gestione del *sistema*$_G$ in generale.

Possiamo quindi compartimentalizzare le varie componenti del sistema in questo modo:

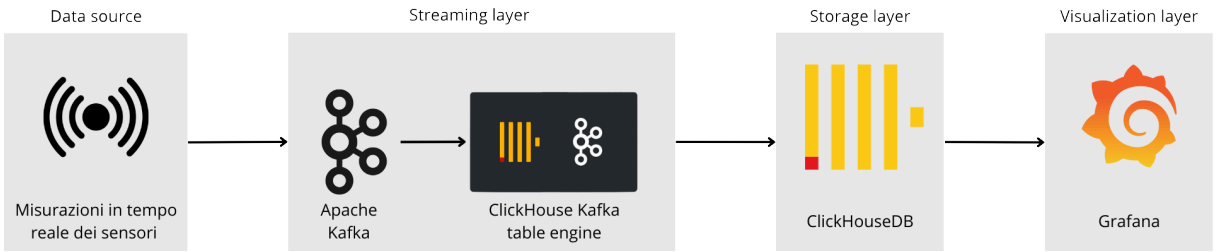

Figure 1: Schema delle componenti del sistema.

- **Data source**: le sorgenti dati sono i sensori IoT sparsi nella città, capaci di inviare messaggi contenenti misurazioni, ad intervalli regolari, mediante protocollo *Kafka*$_G$, allo streaming layer;
- **Streaming layer**: lo streaming layer si occupa di gestire i dati in arrivo in tempo reale, per andarli sistematicamente a persistere nello storage layer. Questo layer è composto da:

  - **Apache Kafka**: che svolgerà il ruolo di *broker*$_G$ dati;
  - **ClickHouse Kafka table engine**: che svolgerà il ruolo di consumatore, al fine di leggere i dati dal server *Kafka*$_G$ per poi persisterli nello storage layer.
- **Storage layer**: si occupa della persistenza dei dati e, grazie alle funzionalità OLAP offerte dal database *ClickHouse*$_G$, di effettuare analisi in tempo reale;
- **Visualization Layer**: composto unicamente da *Grafana*$_G$, questo layer si occupa di visualizzare i dati elaborati, in tempo reale.

3.2 Diagramma del flusso di dati (*data flow diagram*$_G$)

Per illustrare il funzionamento del *sistema*$_G$, abbiamo utilizzato un diagramma di flusso dei dati. Questo diagramma ha permesso di rappresentare in modo chiaro e intuitivo il percorso dei dati attraverso il *sistema*$_G$ e le relative elaborazioni su di essi. Abbiamo quindi identificato le diverse entità coinvolte nel processo e le relazioni tra di esse, fornendo una panoramica dettagliata di come i dati vengono acquisiti, elaborati, archiviati e visualizzati.

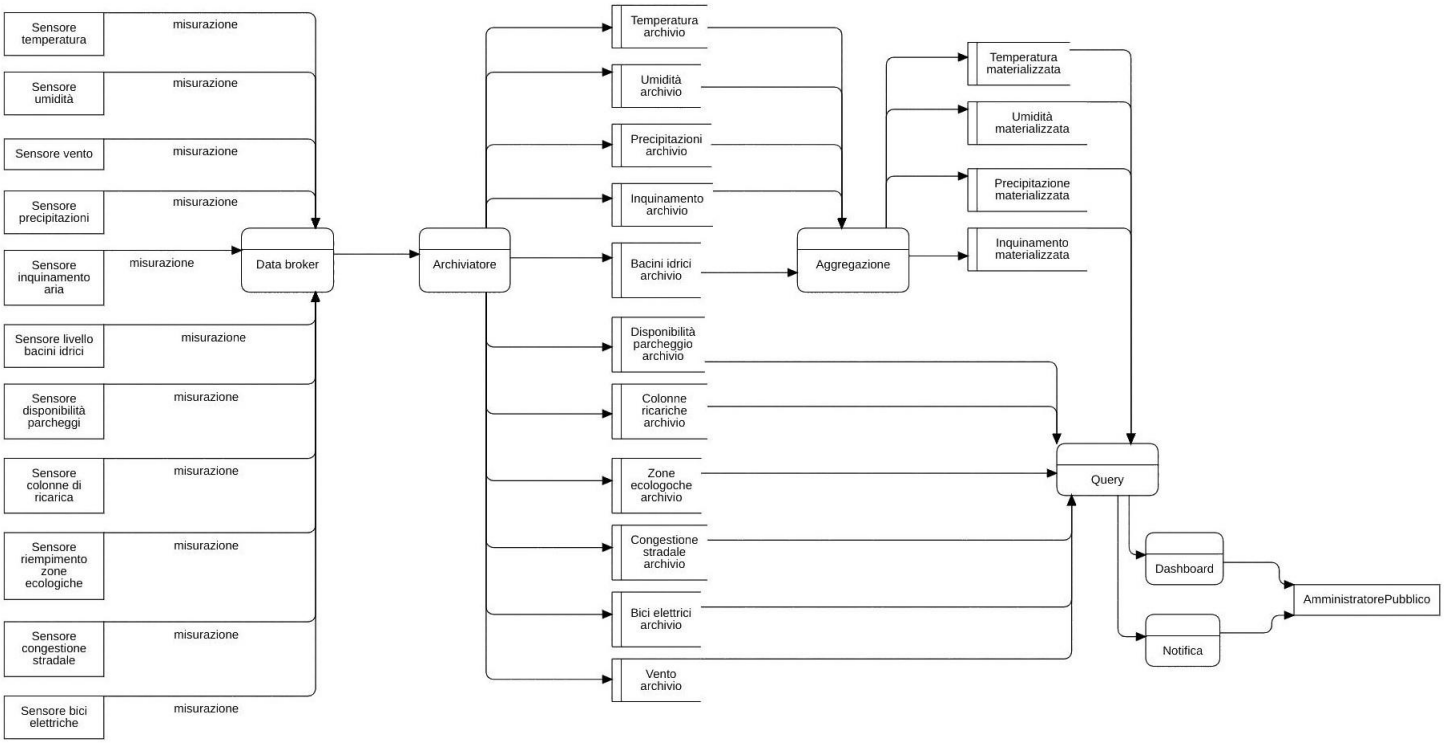

Figure 2: Diagramma data flow relativo al percorso dei dati.

- **Generazione dei dati**: una varietà di simulatori di sensori di dati ambientali e urbanistici (entità esterne) sono utilizzati per misurare una vasta gamma di parametri. Questi simulatori forniscono dati, in modalità continua, relativi a:
  1. Temperatura;
  2. Umidità;
  3. Vento;
  4. Precipitazioni atmosferiche;
  5. Inquinamento atmosferico;
  6. Livello dei bacini idrici;
  7. Disponibilità di parcheggi;
  8. Erogazione delle colonne di ricarica;
  9. Riempimento delle zone ecologiche;

10. Livelli di congestione stradale;
11. Batteria delle biciclette elettriche.

- **Invio al *broker*$_G$ dei dati**: i dati generati dai sensori vengono inviati al *broker*$_G$ dati, in questo contesto *Kafka*$_G$. Quest'ultimo offre un meccanismo di messaggistica distribuita in grado di gestire grandi volumi di dati in tempo reale;
- **Engine interno (archiviatore)**: l'archiviatore, rappresentato dal motore interno "Kafka" di *ClickHouse*$_G$, agisce direttamente come consumatore dei dati provenienti dal *broker*$_G$ dati. Questo avviene tramite la connessione a specifici *topic*$_G$ in *Kafka*$_G$, ognuno associato a un tipo di *sensore*$_G$ distinto. Successivamente, i dati corrispondenti vengono archiviati nelle rispettive tabelle del database;
- **Aggregazione**: i dati corrispondenti a temperatura, umidità, precipitazioni, inquinamento atmosferico e livello dei bacini idrici vengono aggregati in tabelle apposite attraverso l'utilizzo di *materialized views*$_G$. Per ognuno dei tipi di dato citati sopra, vengono aggregati i dati per determinati intervalli di tempo in modo da poter inserire le medie calcolate in tabelle apposite in modo efficiente. Vengono effettuate due aggregazioni distinte per ogni tipo di dato di interesse:
  - Aggregazione per intervalli di 1 minuto; viene utilizzata per mostrare l'andamento in media aritmetica del singolo *sensore*$_G$ di un determinato tipo, utilizzando un intervallo di tempo sufficientemente piccolo per poter analizzare i dati in tempo reale;
  - Aggregazione per intervalli di 5 minuti; viene utilizzata per mostrare l'andamento in media aritmetica di tutti i sensori di un determinato tipo, utilizzando un intervallo di tempo sufficientemente ampio per poter individuare facilmente eventuali trend dei dati;
- **Interrogazioni (query)**: vengono effettuate varie interrogazioni e analisi sui dati memorizzati all'interno delle tabelle;
- **Dashboard**: l'*amministratore pubblico*$_G$ visualizza i dati restituiti in output dalle query ed elaborati attraverso delle *dashboard*$_G$, sulla una piattaforma *Grafana*$_G$;
- **Notifica**: in caso di superamento di determinate soglie pre-impostate, relative ai dati restituiti dalla query, viene inviata una notifica ad un canale appositamente impostato nella piattaforma Discord in modo che possa essere visionata dall'*amministratore pubblico*$_G$.

3.3 Struttura dei container

Abbiamo adottato una struttura basata su container per il nostro sistema, utilizzando Docker e *Docker Compose*$_G$ per gestire l'ambiente di sviluppo e produzione. Questa decisione è stata presa per diversi motivi:

- **Isolamento delle risorse**: utilizzando i container, è possibile isolare ogni componente del *sistema*$_G$, garantendo che le dipendenze siano gestite in modo efficiente e che le modifiche ad un componente non influiscano sugli altri;
- **Portabilità**: i container forniscono un ambiente consistente in cui il software può essere eseguito indipendentemente dal sistema operativo sottostante. Questo consente di distribuire il *sistema*$_G$ su diverse piattaforme senza doversi preoccupare delle differenze di configurazione;
- **Gestione semplificata delle dipendenze**: con *Docker Compose*$_G$, si possono definire facilmente le dipendenze tra i diversi servizi del *sistema*$_G$ e avviare l'intera infrastruttura con un singolo comando. Questo semplifica lo sviluppo e il testing del *sistema*$_G$.

3.3.1 Configurazione dei servizi

Abbiamo definito diversi servizi all'interno del file docker-compose.yml, ognuno dei quali rappresenta una componente critica del *sistema*$_G$. Di seguito sono elencati i principali servizi insieme alla loro configurazione:

3.3.1.1 Servizio Kafka

- Immagine: bitnami/kafka:3.7.0
- Porte esposte: 9093:9093
- Variabili d'ambiente: configurazioni specifiche per *Kafka*$_G$, incluse le porte, gli host e le impostazioni di sicurezza;
- Healthcheck: verifica periodica dello stato di salute di *Kafka*$_G$ per garantire il suo corretto funzionamento.

3.3.1.2 Servizio ClickHouse

- Immagine: clickhouse/clickhouse-server:24-alpine
- Porte esposte: 8123:8123
- Variabili d'ambiente: configurazioni per la creazione del database iniziale e l'autenticazione dell'utente;
- Volume: montaggio di file di configurazione specifici e di script per l'inizializzazione del database.
- Healthcheck: verifica periodica dello stato di salute di *ClickHouse*$_G$.

3.3.1.3 Servizio Grafana

- Immagine: grafana/grafana-oss:10.3.0
- Porte esposte: 3000:3000
- Volume: montaggio di configurazioni e *dashboard*$_G$ personalizzate per *Grafana*$_G$
- Variabili d'ambiente: configurazioni per l'autenticazione e l'installazione di plugin;
- Dipendenze: dipendenza dal servizio *ClickHouse*$_G$ per garantire che *Grafana*$_G$ abbia accesso ai dati.

3.3.1.4 Servizio Simulators

- Build: utilizzo di un Dockerfile personalizzato per costruire il servizio;
- Variabili d'ambiente: configurazioni per il simulatore, incluse le impostazioni del fuso orario e l'host *Kafka*$_G$
- Dipendenze: dipendenza dal servizio *Kafka*$_G$ per garantire una corretta comunicazione.

3.3.2 Profili

Ogni servizio è stato configurato per supportare i profili "dev" e "prod", permettendo una gestione flessibile delle configurazioni in base all'ambiente di deployment.

Questa struttura containerizzata consente di rendere il *sistema*$_G$ modularizzato, scalabile e facilmente gestibile in ambienti di sviluppo e produzione.

3.4 Database

Lo scopo del database è quello di memorizzare i dati provenienti dai sensori, in modo da poterli analizzare e visualizzare in seguito. I dati di un *sensore*$_G$ vengono acquisiti tramite un *topic*$_G$ *Kafka*$_G$, associato ad un tipo di *sensore*$_G$, e poi memorizzati in apposite tabelle.

3.4.1 Struttura

3.4.1.1 Tabella di accodamento

Ad ogni tipo di *sensore*$_G$ viene assegnata una tabella con il nome definito come: *tipo*+-topic+kafka. Questa conterrà, per ogni messaggio proveniente dal *topic*$_G$ dedicato, un record contenente una stringa con all'interno tutti i dati grezzi provenienti dai sensori di quel tipo, che sono in formato *JSON*$_G$ e contengono, oltre alla rilevazione con relativo timestamp, anche il nome e la tipologia del *sensore*$_G$ e le sue coordinate geografiche. Le tabelle di questo tipo vengono popolate tramite il "ClickHouse Kafka table engine", che si occupa di leggere la stringa di dati, dal *topic*$_G$ di *Kafka*$_G$ e di "iniettarla" all'interno della tabella; per compiere quest'operazione è necessario specificare il motore di archiviazione "Kafka".

3.4.1.2 Tabella *time series*$_G$

Per ogni tipo di *sensore*$_G$ viene creata una tabella *time series*$_G$, con il nome definito come: *tipo*. Il suo scopo è gestire in modo efficiente i dati di tipo *time series*$_G$, tramite il motore di archiviazione MergeTree. Queste tabelle vengono popolate tramite delle *materialized views*$_G$ a partire dalle tabelle di accodamento; vengono inoltre ordinate per *sensore*$_G$ e per timestamp.

3.4.1.3 Tabella dati aggregati

Questo tipo di tabelle aggrega i dati provenienti dalle tabelle *time series*$_G$ in modo da poter effettuare analisi su intervalli temporali specifici. Ad esempio:

- Media aritmetica dell'andamento di un singolo *sensore*$_G$ di un determinato tipo con campionamenti ogni minuto;
- Media aritmetica dell'andamento di tutti i sensori di un determinato tipo con campionamenti ogni cinque minuti.

Per il loro popolamento si utilizzano le *materialized views*$_G$, che permettono di inserire i dati risultanti da una query, tra i quali la media.

3.4.1.4 Motori di archiviazione

ClickHouse$_G$ offre diversi motori di archiviazione, ognuno con caratteristiche specifiche. Per il nostro caso d'uso, facciamo uso dei seguenti:

- **Kafka**: permette di leggere la stringa contenente i dati in formato *JSON*$_G$ dal *broker*$_G$ dati, e di iniettarla all'interno delle tabelle di accodamento;
- **MergeTree**: particolarmente adatto per la gestione di dati di tipo *time series*$_G$, in quanto permette di effettuare operazioni di inserimento ed eliminazione in modo efficiente, e di effettuare query su intervalli di tempo specifici.

3.4.2 Schema

Il database è caratterizzato da un determinato schema per ogni tipo di *sensore*$_G$; di seguito vengono illustrate, tramite un esempio, le due tipologie di schema utilizzate a seconda del tipo di *sensore*$_G$.

3.4.2.1 Sensore per il vento

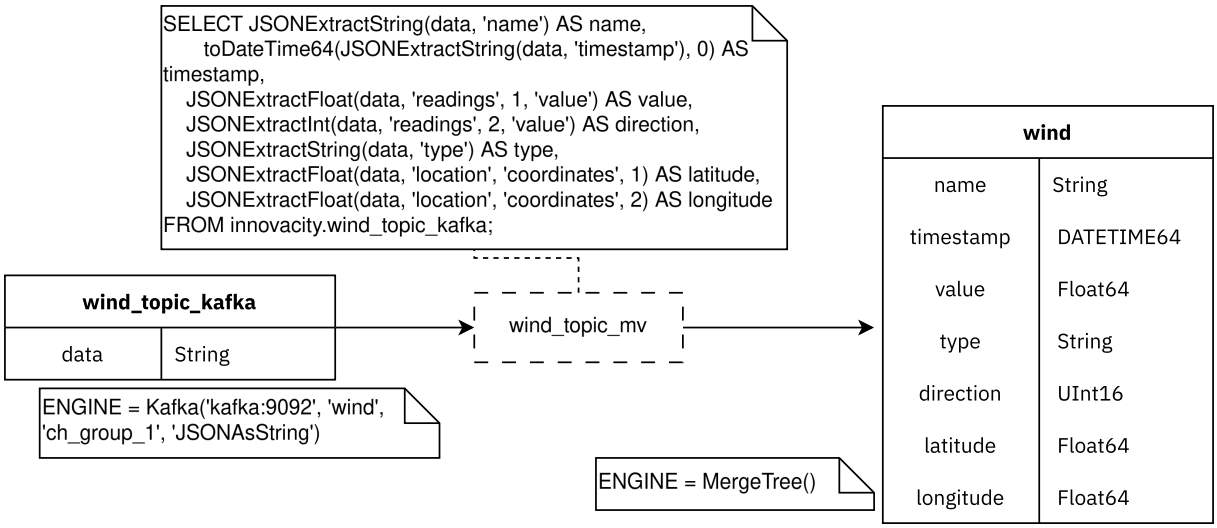

Figure 3: Schema delle tabelle per il *sensore*$_G$ del vento

Ad esempio la funzione `JSONExtractString` è una funzione di *ClickHouse*$_G$ che permette di estrarre una stringa da un campo *JSON*$_G$; in questo caso, viene utilizzata per estrarre il campo “name” dal campo “data” del messaggio *JSON*$_G$. Lo stesso vale per le altre funzioni utilizzate: `JSONExtractFloat`, e `JSONExtractInt` e per i rimanenti campi.

3.4.2.2 Sensore pluviometrico

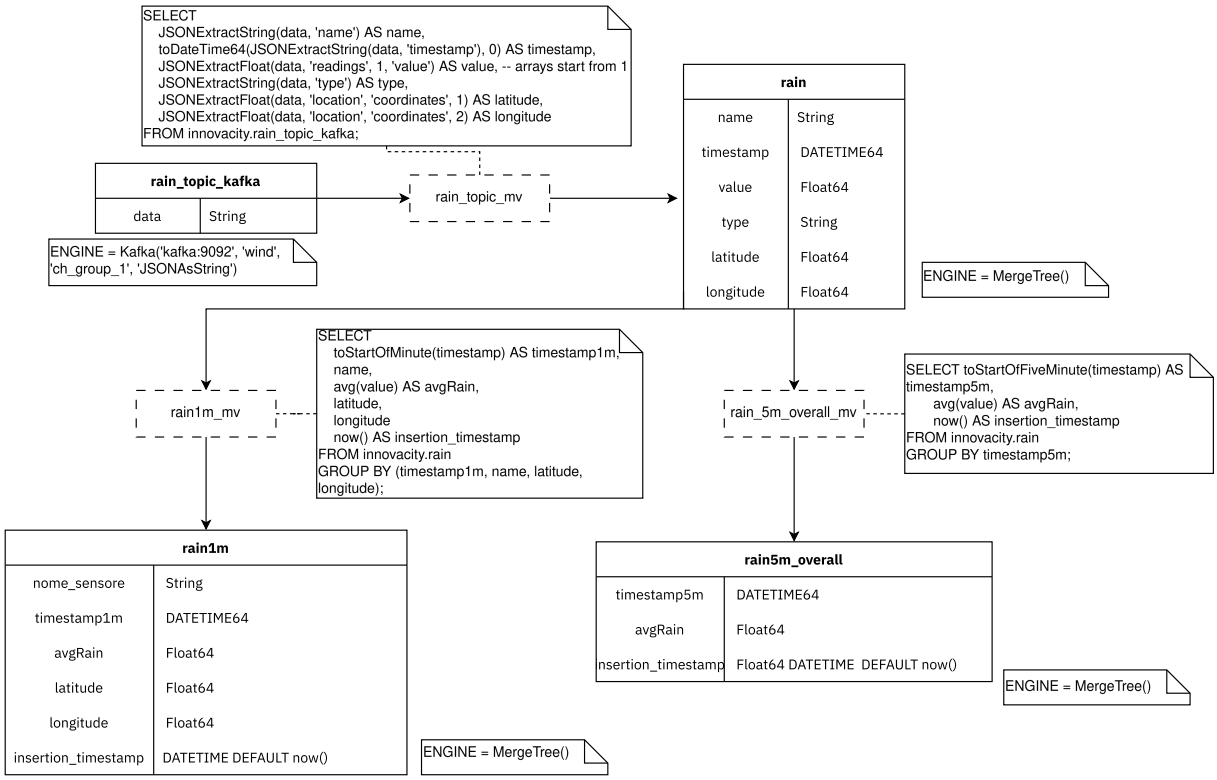

Figure 4: Schema delle tabelle per il *sensore*$_G$ pluviometrico

Questo tipo di schema presenta in più, rispetto a quello precedente due tabelle per dati aggregati. La tabella “rain1m” contiene la media aritmetica, ottenuta aggregando i dati della pioggia ogni minuto per ciascun *sensore*$_G$; invece, la tabella “rain5m_overall” contiene la media aritmetica, ottenuta aggregando i dati della pioggia ogni cinque minuti per tutti i sensori che monitorano le precipitazioni. Questi tipi di tabelle vengono popolati tramite delle *materialized views*$_G$.

3.5 *Architettura*$_G$ dei simulatori

Nonostante i simulatori non siano ufficialmente considerati parte integrante del prodotto dalla Proponente, il nostro team, nell'ambito del progetto didattico, ha scelto di dedicare alcune risorse alla progettazione di questa componente.

Nei paragrafi successivi viene mostrata l'*architettura*$_G$ individuata, tramite l'utilizzo di Diagrammi delle Classi e relative rapide descrizioni. Inoltre vengono motivati i *design pattern*$_G$ individuati e le decisioni progettuali rilevanti. Successivamente, per ogni classe vengono illustrati metodi e attributi.

3.5.1 Struttura generale

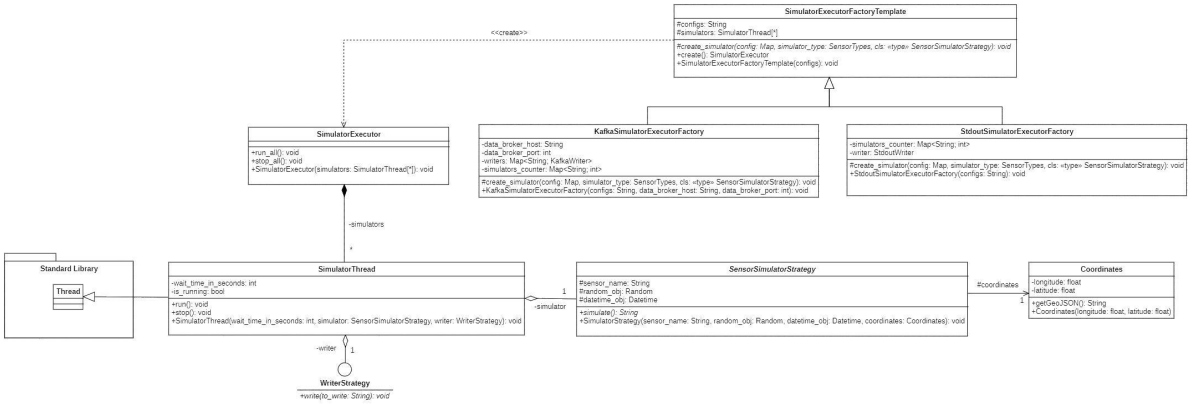

Figure 5: Diagramma delle classi 2

La classe *SimulatorExecutorFactoryTemplate* è implementazione del *design pattern*$_G$ *Template*, si occupa di invocare la creazione dei singoli simulatori per poi restituire un *SimulatorExecutor* che avrà il compito di orchestrarli. Il metodo utilizzato per la creazione del singolo simulatore viene ereditato ed implementato diversamente, a seconda del tipo di *Writer* che si vuole utilizzare, da parte delle classi *KafkaSimulatorExecutorFactory* e *StdoutSimulatorExecutorFactory*; entrambe sono implementazioni del *design pattern*$_G$ *Factory*, in quanto si occupano della creazione vera e propria dei simulatori a partire da un file di configurazione che viene passato tramite costruzione. I simulatori sono istanze delle classe *SimulatorThread*: tale classe eredita dalla classe *Thread* della Standard Library, in modo tale che l'esecuzione dei simulatori possa essere parallela.

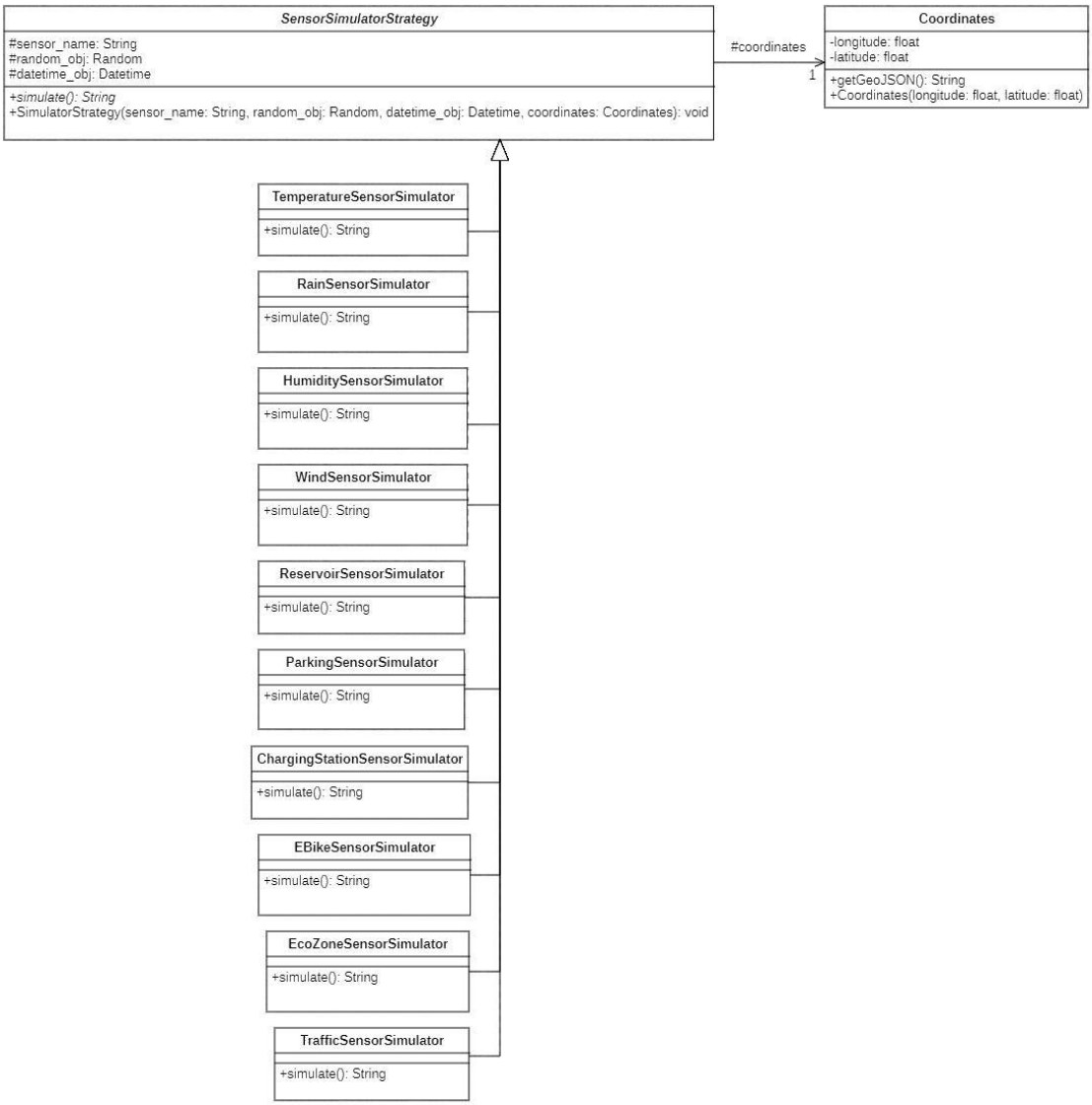

Figure 6: Diagramma delle classi 2

La classe *SensorSimulatorStrategy* è realizzazione del design pattern *Strategy*, dove ogni strategia rappresenta una tipologia di *sensore*$_G$ simulato differente. Al fine di garantire la possibilità di effettuare unit-testing sul comportamento dei simulatori di sensori tale classe riceve tramite costruttore un oggetto di tipo *Random* e un oggetto di tipo *Datetime*. Tali strategie verranno poi assegnate ad un *SimulatorThread*.

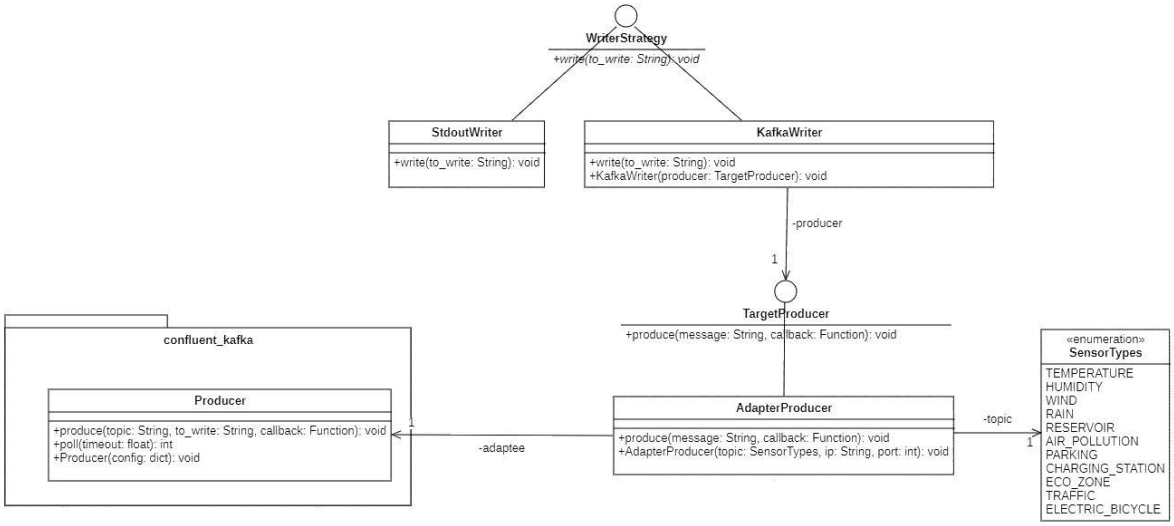

Figure 7: Diagramma delle classi 3

Anche la classe *Writer* realizza il design pattern *Strategy*. Sono state progettate due strategie, la prima, (*KafkaWriter*), atta a permettere al simulatore di inviare i messaggi contenenti i dati della rilevazione a *Kafka*$_G$, mentre la seconda atta a permettere al simulatore di stampare i risultati su terminale al fine di poterne testare il comportamento. Inoltre l'applicazione di tale *design pattern*$_G$ potrebbe consentire di realizzare il componente di scrittura anche per eventuali *broker*$_G$ dati alternativi, nel momento in cui ce ne sia il bisogno. Nello specifico, la classe *KafkaWriter* realizza il suo scopo tramite l'impiego del *design pattern*$_G$ *Adapter*, nella sua variante *Object Adapter*. Viene infatti fatto utilizzo della classe *Producer* della liberia *confluent_kafka*. Dato che tale classe potrebbe essere soggetta a variazioni non controllabili da noi, si è deciso di utilizzare tale pattern per permettere di rispondere prontamente a tali cambiamenti, spostandone la complessità proprio nell'adapter.

3.5.2 Classi: metodi e attributi

Si procede alla descrizione di classi e interfacce progettate dal team. Non vengono menzionati i costruttori.

3.5.2.1 SimulatorExecutorFactoryTemplate (Classe)

3.5.2.1.1 Attributi

- **configs: String [Protetto]**: contenuto del file di configurazione dei simulatori;
- **simulators: SimulatorThread[*] [Protetto]**: lista di oggetti *SimulatorThread*.

3.5.2.1.2 Metodi

- **create_simulator(config: Map, simulator_type: SensorTypes, cls: «type» SensorSimulatorStrategy): void [Protetto]**: metodo astratto, implementato dalle classi *KafkaSimulatorExecutorFactory* e *StdoutSimulatorExecutorFactory*;
- **create(): SimulatorExecutor [Pubblico]**: dopo aver invocato la creazione dei singoli simulatori, si occupa di costruire un oggetto della classe *SimulatorExecutor* inserendoli al suo interno.

3.5.2.2 KafkaSimulatorExecutorFactory (Classe)

3.5.2.2.1 Attributi

- **data_broker_host: String [Privato]**: indirizzo ip della macchina che ospita il message *broker*$_G$
- **data_broker_port: int [Privato]**: porta della macchina che ospita il message *broker*$_G$
- **writers: Map<String, KafkaWriter> [Privato]**: dizionario con chiavi di tipo String e valori di tipo *KafkaWriter*;
- **simulators_counter: Map<String, int> [Privato]**: dizionario con chiavi di tipo String e valori di tipo int, per fare in modo che al nome del simulatore sia associato un valore numerico.

3.5.2.2.2 Metodi
- **create_simulator(config: Dict, simulator_type: SensorTypes, cls: «type» SensorSimulatorStrategy): void [Protetto]**: utilizza il contenuto del file di configurazione passato tramite costruzione per costruire gli oggetti di tipo *SimulatorThread* che rappresentano i simulatori richiesti.

3.5.2.3 StdoutSimulatorExecutorFactory (Classe)

3.5.2.3.1 Attributi
- **simulators_counter: Map<String, int> [Privato]**: dizionario per fare in modo che al nome del simulatore sia associato un valore numerico;
- **writer: StdoutWriter [Privato]**: oggetto della classe *StdoutWriter*.

3.5.2.3.2 Metodi
- **create_simulator(config: Map, simulator_type: SensorTypes, cls: «type» SensorSimulatorStrategy): void [Protetto]**: utilizza il contenuto del file di configurazione passato tramite costruzione per costruire gli oggetti di tipo *SimulatorThread* che rappresentano i simulatori richiesti.

3.5.2.4 SimulatorExecutor (Classe)

3.5.2.4.1 Attributi
- **simulators: SimulatorThread[*] [Privato]**: lista di oggetti *SimulatorThread*.

3.5.2.4.2 Metodi
- **run_all(): void [Pubblico]**: metodo che avvia la simulazione di tutti i *SimulatorThread*;
- **stop_all(): void [Pubblico]**: metodo che arresta la simulazione di tutti i *SimulatorThread*.

3.5.2.5 SimulatorThread (Classe)

3.5.2.5.1 Attributi
- **wait_time_in_seconds: int [Privato]**: intervallo di tempo (in secondi) che trascorre tra una simulazione e quella successiva;
- **is_running: bool [Privato]**: attributo che indica se al momento il simulatore sta producendo dati;
- **simulator: SensorSimulatorStrategy [Privato]**: oggetto della classe *SensorSimulatorStrategy*;
- **writer: WriterStrategy [Privato]**: oggetto della classe *WriterStrategy*.

3.5.2.5.2 Metodi
- **run(): void[Pubblico]**: metodo che avvia la simulazione del singolo *SimulatorThread*;
- **stop(): void[Pubblico]**: metodo che arresta la simulazione del singolo *SimulatorThread*.

3.5.2.6 SensorSimulatorStrategy (Classe)

3.5.2.6.1 Attributi
- **sensor_name: String [Protetto]**: nome identificativo del sensore;
- **random_obj: Random [Protetto]**: oggetto della classe Random della libreria Standard di *Python*$_G$;
- **datetime_obj: Datetime [Protetto]**: oggetto della classe Datetime della libreria Standard di *Python*$_G$;
- **coordinates: Coordinates [Protetto]**: oggetto della classe Coordinates.

3.5.2.6.2 Metodi
- **simulate(): String [Pubblico]**: metodo astratto, implementato dalle classi che definiscono il comportamento dei singoli simulatori.

3.5.2.7 Coordinates (Classe)

3.5.2.7.1 Attributi
- **longitude: float [Privato]**: longitudine geografica,
- **latitude: float [Privato]**: latitudine geografica.

3.5.2.7.2 Metodi
- **getGeoJSON(): String [Pubblico]**: restituisce le coordinate geografiche nel formato GeoJSON.

3.5.2.8 SensorTypes (Class)

3.5.2.8.1 Attributi
- TEMPERATURE: String [Public];
- HUMIDITY: String [Public];
- WIND: String [Public];
- RAIN: String [Public];
- RESERVOIR: String [Public];
- AIR_POLLUTION: String [Public];
- PARKING: String [Public];
- CHARGING_STATION: String [Public];
- ECO_ZONE: String [Public];
- TRAFFIC: String [Public];
- ELECTRIC_BICYCLE: String [Public].

3.5.2.9 WriterStrategy (interfaccia)

3.5.2.9.1 Metodi
- **write(to_write: String): void [Pubblico]**: metodo astratto, implementato dalle classi *StdOutWriter* e *KafkaWriter*.

3.5.2.10 StdOutWriter (Classe)

3.5.2.10.1 Metodi
- **write(to_write: String): void [Pubblico]**: scrive il messaggio contenuto nella stringa passata come parametro su standard output.

3.5.2.11 KafkaWriter (Classe)

3.5.2.11.1 Attributi

- **producer: TargetProducer [Privato]**: oggetto della classe *TargetProducer*.

3.5.2.11.2 Metodi

- **write(to_write: String): void [Pubblico]**: invoca la scrittura del messaggio contenuto nella stringa passata come parametro sul *topic*$_G$ corrispondente alla tipologia del simulatore nel *broker*$_G$.

3.5.2.12 TargetProducer (Interfaccia)

3.5.2.12.1 Metodi

- **produce(message: String, callback: Function): void [Pubblico]**: metodo astratto, utilizzato dal client per inviare il messaggio a *Kafka*$_G$.

3.5.2.13 AdapterProducer (Interfaccia)

3.5.2.13.1 Attributi

- **adaptee: Producer [Privato]**: oggetto della classe *Producer* della libreria *Confluent Kafka*;
- **topic: SensorTypes [Privato]**: oggetto della classe *SensorTypes*, rappresenta il *topic*$_G$ all'interno del *broker*$_G$ nel quale viene inserito il messaggio da scrivere.

3.5.2.13.2 Metodi

- **produce(message: String, callback: Function): void [Pubblico]**: metodo che richiama il metodo *produce()* dell'oggetto *adaptee* di tipo *Producer* della libreria *Confluent Kafka*.

3.6 Messaggi ed Eventi

3.6.1 Kafka Topics

I *topic*$_G$ in *Kafka*$_G$ possono essere visti come le tabelle di un database, servono per separare logicamente diversi tipi di messaggi o eventi che vengono inseriti nel *sistema*$_G$. In questo caso vengono utilizzati per separare i messaggi provenienti dai diversi tipi di *sensore*$_G$; questo permette poi di andare a creare all'interno di ClickHouse delle "tabelle consumatrici" che prendono i dati in automatico, grazie al fatto che avendo separati logicamente i *topic*$_G$, i messaggi all'interno di ognuno di essi hanno tutti lo stesso formato.

3.6.2 Struttura dei messaggi

La struttura di un messaggio o evento, descritta in JSON sarà la seguente:

```
{
    "type": "SomethingSimulator",
    "timestamp": "2024-01-10 00:00:00",
    "readings":[
        {
            "type": "Speed",
            "value": "00.00"
        },
        {
            "type": "Direction",
            "value": "270"
        }
    ],
    "nome": "Significant Name",
    "location":{
        "type": "Point",
        "coordinates": [00.000000, 00.000000]
```

```
    }
}
```

Gli oggetti all'interno di "readings" sono solamente esempi, è possibile inserirne uno solo, come ad esempio per i sensori di temperatura, che avranno solamente la lettura omonima, oppure, come nel caso del *sensore*$_G$ del vento averne due: la velocità e la direzione.

3.7 Configurazione visualizzazione e sistema di allerta

3.7.1 Dashboard

La visualizzazione dei dati attraverso *pannelli*$_G$ si compone delle seguenti *dashboard*$_G$ in *Grafana*$_G$. Le configurazioni delle *dashboard*$_G$ vengono salvate in formato *JSON*$_G$, il che permette di svilupparle e manutenerle agevolmente. I dati per la visualizzazione sono prelevati dalle tabelle di *ClickHouse*$_G$ tramite query sulle tabelle *time series*$_G$ e di dati aggregati. Di seguito vengono descritte le *dashboard*$_G$ realizzate all'interno del visualization layer:

- **Ambientale**: visualizzazione dei dati relativi a temperatura, umidità, inquinamento dell'aria, vento e precipitazioni atmosferiche, tramite grafici a linee, mappe e indicatori numerici; include, inoltre, una mappa che illustra la posizione dei sensori e il loro tipo, riconoscibile grazie ad un colore apposito;
- **Urbanistica**: visualizzazione dei dati relativi a parcheggi, colonne di ricarica, zone ecologiche, livello di congestione stradale e batteria delle biciclette elettriche, tramite mappe ed indicatori numerici;
- **Dati grezzi**: visualizzazione dei dati grezzi provenienti dai sensori, tramite una tabella che mostra i dati in tempo reale, con la possibilità di filtrare per specifico *sensore*$_G$ e per tipo;
- **Superamento soglie**: visualizzazione dei dati relativi al superamento delle soglie di allerta per i diversi tipi di sensori; i dati superanti le soglie pre-impostate vengono visualizzati tramite una tabella che mostra il nome del *sensore*$_G$ che ha effettuato la misurazione, il valore rilevato e il timestamp corrispondente.

3.7.2 Sistema di notifica

Per soddisfare i requisiti di ricezione di notifiche relativi a dati, che abbiano superato un determinata soglia, il team decide di utilizzare il sistema di allerting integrato in *Grafana*$_G$. Questo sistema si rivela efficace grazie alla sua capacità di integrarsi con diverse piattaforme di messaggistica in tempo reale, tra le quali Discord.

Una volta che un'allerta viene rilevata, passa attraverso tre stati distinti:

- in attesa (pending): l'allerta è stata innescata ma la sua conferma è ancora in corso. Questo stato è tipicamente utilizzato per regolare l'invio delle notifiche, assicurandosi che l'allerta sia stabile prima di comunicare il problema alla sua origine;
- attiva (firing): l'allerta è stata confermata e la condizione critica è stata verificata, di conseguenza vengono inviate le notifiche ai canali configurati;
- ok: la situazione che ha innescato l'allerta è tornata alla normalità.

Nella realizzazione del componente di notifica il team ha optato per mantenerlo il più semplice possibile, evitando lo stazionamento dell'allerta nella fase di "pending", in modo tale che ogni allerta venga notificata appena rilevata. Inoltre, è risultato preferibile la rimozione delle notifiche relative al rientro nella condizione di "ok".

Ogni allerta viene definita all'interno di un "alert group" a sé stante, in modo tale che ognuna sia eseguita in contemporanea alle altre. Ogni regola verifica, a intervalli regolari di 5 minuti, se il superamento della soglia impostata si sia verificato nell'intervallo dei 5 minuti precedenti; inoltre, trattiene il valore massimo tra quelli superanti la soglia, per ogni singolo *sensore*$_G$. *Grafana*$_G$ permette inoltre di inserire ulteriori configurazioni relative propriamente alle modalità di invio e alla personalizzazione delle notifiche nelle sezioni "Notification Policy" e "Contact Points", garantendo un maggiore controllo e una maggiore flessibilità nella gestione

delle notifiche. Sia le regole di allerta, le configurazioni dei canali di notifica, che le “Notification Policy”, possono essere impostate tramite l’interfaccia grafica; successivamente, possono essere esportate in vari formati e inserite in file di configurazione appropriati, all’interno della directory /provisoning/alerting per garantire la persistenza. Nel nostro caso si è scelto di utilizzare il formato *JSON*$_G$.

Relativamente a questi elementi di configurazione, per garantire che il sistema di notifica rimanga semplice e intuitivo, ci si limita a configurare il canale di notifica e il formato delle notifiche stesse, oltre alle configurazioni relative alla loro frequenza. Si è deciso di utilizzare Discord come canale di notifica preferenziale: nello specifico per la sua configurazione è necessario inserire il corrispondente webhook URL.

4 Tracciamento dei requisiti

In questa sezione si va a mostrare, secondo quanto riportato dal documento *Norme di Progetto v2.0*, la soddisfazione dei singoli requisiti presenti, in base al tipo previsto e opportunamente classificato sotto.

4.1 Tabella dei requisiti soddisfatti

Si vuole riportare ciascun requisito mediante il corrispondente codice, utilizzando le seguenti sigle, le quali indicano:

- RO - Requisito Obbligatorio;
- RD - Requisito Desiderabile;
- RP - Requisito Opzionale.

Rispetto alla stessa tabella ritrovabile nel documento *Analisi dei Requisiti v2.0*, qui è presente una colonna *Stato* indicante la soddisfazione di tale requisito.

Codice	Descrizione	Stato
ROF1	L'utente deve poter accedere all'applicazione senza dover effettuare l'autenticazione.	Soddisfatto
ROF2	L'utente deve poter visualizzare un menù di selezione delle *dashboard*$_G$, che permetta di selezionare tra Dati grezzi, Ambientale, Urbanistica e Superamento soglie.	Soddisfatto
ROF3	L'utente deve poter visualizzare una *dashboard*$_G$ generale relativa ai dati grezzi.	Soddisfatto
ROF4	L'utente deve poter visualizzare, in forma tabellare, i dati grezzi inviati da tutti i sensori con il nome del *sensore*$_G$, la tipologia del *sensore*$_G$, il timestamp della rilevazione e il valore della misurazione (se composta da più dati, tutti i valori sono elencati nella colonna corrispondente), all'interno della *dashboard*$_G$ relativa ai dati grezzi.	Soddisfatto
ROF5	L'utente deve poter monitorare i dati provenienti dai sensori relativi ai dati ambientali in una *dashboard*$_G$ apposita.	Soddisfatto
ROF6	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante l'andamento in media aritmetica della temperatura, espressa in gradi Celsius, per ciascun *sensore*$_G$, aggregando i dati per intervalli di 1 minuto, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
ROF7	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante l'andamento in media aritmetica della temperatura, espressa in gradi Celsius, di tutti i sensori, aggregando i dati per intervalli di 5 minuti, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
ROF8	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante l'andamento in media aritmetica della percentuale d'umidità, per ciascun *sensore*$_G$, aggregando i dati per intervalli di 1 minuto, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto

ROF9	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante l'andamento in media aritmetica della percentuale d'umidità, di tutti i sensori, aggregando i dati per intervalli di 5 minuti, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
RDF10	L'utente deve poter visualizzare un *pannello*$_G$ contenente una mappa che evidenzi la direzione del vento, mediante frecce collocate nelle coordinate geografiche del *sensore*$_G$, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
ROF11	L'utente deve poter visualizzare un *pannello*$_G$ contenente una tabella la quale riporta l'ultima velocità del vento, espressa in chilometri all'ora (km/h), e la sua direzione, espressa in gradi (con gli 0° a Est e i 180° a Ovest), per ciascun *sensore*$_G$, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
ROF12	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante l'andamento in media aritmetica dell'intensità delle precipitazioni, espresse in millimetri all'ora (mm/h), per ciascun *sensore*$_G$, aggregando i dati per intervalli di 1 minuto, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
ROF13	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante l'andamento in media aritmetica dell'intensità delle precipitazioni, espresse in millimetri all'ora (mm/h), di tutti i sensori, aggregando i dati per intervalli di 5 minuti, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
RDF14	L'utente deve poter visualizzare un *pannello*$_G$ contenente un indice numerico rappresentante l'intensità media delle precipitazioni, espressa in millimetri all'ora (mm/h), nell'intervallo di tempo impostato all'interno della *dashboard*$_G$, facendo la media dei dati raccolti tra tutti i sensori, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
ROF15	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante l'andamento in media aritmetica del livello di polveri sottili nell'aria, espressi in $\mu g/m^3$ (*PM10*$_G$), per ciascun *sensore*$_G$, aggregando i dati per intervalli di 1 minuto, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
ROF16	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante l'andamento in media aritmetica del livello di polveri sottili nell'aria, espressi in $\mu g/m^3$ (*PM10*$_G$), di tutti i sensori, aggregando i dati per intervalli di 5 minuti, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
RDF17	L'utente deve poter visualizzare un *pannello*$_G$ contenente un indice numerico, che esprime il livello medio di polveri sottili nell'aria, espresso in $\mu g/m^3$ (*PM10*$_G$), nell'ultimo minuto, facendo	Soddisfatto

	la media dei dati raccolti tra tutti i sensori, nella *dashboard*$_G$ relativa ai dati ambientali.	
ROF18	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante l'andamento in media aritmetica della percentuale di riempimento dei bacini idrici, per ciascun *sensore*$_G$, aggregando i dati per intervalli di 1 minuto, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
ROF19	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante l'andamento in media aritmetica della percentuale di riempimento dei bacini idrici, di tutti i sensori, aggregando i dati per intervalli di 5 minuti, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
RDF20	L'utente deve poter visualizzare un *pannello*$_G$ contenente un indice numerico, che esprime la temperatura media, espressa in gradi Celsius, nell'intervallo di tempo impostato all'interno della *dashboard*$_G$, facendo la media dei dati raccolti tra tutti i sensori, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
RDF21	L'utente deve poter visualizzare un *pannello*$_G$ contenente un indice numerico, che esprime il livello massimo di polveri sottili nell'aria, espresso in $\mu g/m^3$ (*PM10*$_G$), negli ultimi 5 minuti, tra i dati registrati da tutti i sensori, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
ROF22	L'utente deve poter visualizzare un *pannello*$_G$ contenente una mappa che mostri le posizioni dei sensori che monitorano i dati ambientali, mediante icone colorate in base al tipo di *sensore*$_G$, nella *dashboard*$_G$ relativa ai dati ambientali.	Soddisfatto
ROF23	L'utente deve poter monitorare i dati provenienti dai sensori relativi ai dati urbanistici in una *dashboard*$_G$ apposita.	Soddisfatto
ROF24	L'utente deve poter visualizzare un *pannello*$_G$ contenente una mappa che evidenzi il numero di posti liberi nei vari parcheggi, mediante indicatori numerici posti nelle coordinate geografiche del *sensore*$_G$, nella *dashboard*$_G$ relativa ai dati urbanistici.	Soddisfatto
ROF25	L'utente deve poter visualizzare un *pannello*$_G$ contenente icone poste nelle coordinate geografiche dei sensori che indichino la disponibilità delle colonne di ricarica, nella *dashboard*$_G$ relativa ai dati urbanistici.	Soddisfatto
RDF26	L'utente deve poter visualizzare un *pannello*$_G$ contenente una tabella la quale riporta l'erogazione delle colonne di ricarica per auto, espressa in chiloWatt all'ora (kWh), controllata da ciascun *sensore*$_G$, nella *dashboard*$_G$ relativa ai dati urbanistici.	Soddisfatto
ROF27	L'utente deve poter visualizzare un *pannello*$_G$ contenente una mappa che evidenzi lo stato di congestione delle strade, mediante gli stati "LOW", "MEDIUM", "HIGH", "BLOCKED", posti nelle coordinate geografiche dei sensori che le monitorano, nella *dashboard*$_G$ relativa ai dati urbanistici.	Soddisfatto

ROF28	L'utente deve poter visualizzare un *pannello*$_G$ contenente una mappa che mostri la posizione delle biciclette elettriche controllate, in tempo reale, mediante degli indicatori numerici, indicanti la percentuale della batteria, posizionati nelle coordinate geografiche del mezzo, nella *dashboard*$_G$ relativa ai dati urbanistici.	Soddisfatto
ROF29	L'utente deve poter visualizzare un *pannello*$_G$ contenente una mappa che mostri la percentuale di riempimento delle zone ecologiche, mediante degli indicatori percentuali, posizionati nelle coordinate geografiche della zona, nella *dashboard*$_G$ relativa ai dati urbanistici.	Soddisfatto
RDF30	L'utente deve poter monitorare i dati superanti le soglie impostate nel *sistema*$_G$, in una *dashboard*$_G$ apposita.	Soddisfatto
RPF31	L'utente deve poter visualizzare un *pannello*$_G$ contenente una tabella che mostri i *dati anomali*$_G$, il *sensore*$_G$ che li ha rilevati e il timestamp del rilevamento, nella *dashboard*$_G$ relativa ai dati superanti le soglie.	Soddisfatto
RDF32	L'utente deve poter visualizzare un *pannello*$_G$ contenente una tabella che mostri i dati i cui valori superano la soglia dei 40° Celsius (40°C) di temperatura, il *sensore*$_G$ che li ha rilevati e il timestamp del rilevamento, nella *dashboard*$_G$ relativa ai dati superanti le soglie.	Soddisfatto
RDF33	L'utente deve poter visualizzare un *pannello*$_G$ contenente una tabella che mostri i dati i cui valori superano la soglia dei 50 millimetri di pioggia all'ora (50 mm/h), il *sensore*$_G$ che li ha rilevati e il timestamp del rilevamento, nella *dashboard*$_G$ relativa ai dati superanti le soglie.	Soddisfatto
RDF34	L'utente deve poter visualizzare un *pannello*$_G$ contenente una tabella che mostri i dati i cui valori superano la soglia di 80µg su metro cubo ($80\mu g/m^3$) di inquinamento dell'aria, il *sensore*$_G$ che li ha rilevati e il timestamp del rilevamento, nella *dashboard*$_G$ relativa ai dati superanti le soglie.	Soddisfatto
RDF35	L'utente deve poter visualizzare un *pannello*$_G$ contenente una tabella che mostri i dati i cui valori superano la soglia del 70% di capienza di un bacino idrico, il *sensore*$_G$ che li ha rilevati e il timestamp del rilevamento, nella *dashboard*$_G$ relativa ai dati superanti le soglie.	Soddisfatto
RDF36	L'utente deve poter visualizzare un *pannello*$_G$ contenente una tabella che mostri i dati i cui valori superano la soglia dell'80% di capienza di una zona ecologica, il *sensore*$_G$ che li ha rilevati e il timestamp del rilevamento, nella *dashboard*$_G$ relativa ai dati superanti le soglie.	Soddisfatto
RDF37	L'utente deve poter visualizzare delle notifiche che denotano il superamento di una soglia impostata.	Soddisfatto

RDF38	L'utente deve poter visualizzare una notifica relativa alla temperatura che denota il superamento dei 40° Celsius (40°C).	Soddisfatto
RDF39	L'utente deve poter visualizzare una notifica relativa alle precipitazioni che denota il superamento dei 50 millimetri di pioggia all'ora (50 mm/h).	Soddisfatto
RDF40	L'utente deve poter visualizzare una notifica relativa all'inquinamento dell'aria che denota il superamento di 80µg su metro cubo ($80\mu g/m^3$).	Soddisfatto
RDF41	L'utente deve poter visualizzare una notifica relativa ai bacini idrici che denota il superamento del 70% della capienza di un particolare bacino.	Soddisfatto
RDF42	L'utente deve poter visualizzare una notifica relativa alle zone ecologiche che denota il superamento dell'80% della capienza di una particolare zona ecologica.	Soddisfatto
ROF43	L'utente deve poter filtrare i dati, visualizzati all'interno di un grafico di tipo *time series*$_G$, in base ad un sottoinsieme selezionato di sensori.	Soddisfatto
ROF44	L'utente deve poter filtrare i dati, visualizzati all'interno di una tabella, in base ad un sotto-insieme di sensori, selezionandone la tipologia di interesse.	Soddisfatto
ROF45	L'utente deve poter filtrare i dati, visualizzati all'interno di una tabella, in base ad un sotto-insieme di sensori, selezionando i nomi dei sensori di interesse.	Soddisfatto
ROF46	L'utente deve poter filtrare i dati in base ad un intervallo temporale, mostrando quindi nella *dashboard*$_G$ d'interesse, solamente i dati aventi un timestamp in tale intervallo.	Soddisfatto
RDF47	Nei *pannelli*$_G$ con tabelle, l'utente deve poter ordinare i dati in base a tutti i campi presenti, sia in ordine crescente che decrescente.	Soddisfatto
RDF48	L'utente deve poter modificare il layout della *dashboard*$_G$ visualizzata, agendo sullo spostamento dei *pannelli*$_G$.	Non soddisfatto
RDF49	L'utente deve poter modificare il layout della *dashboard*$_G$ visualizzata, agendo sul ridimensionamento dei *pannelli*$_G$.	Non soddisfatto
ROF50	L'utente deve poter visualizzare un messaggio di errore, qualora il *sistema*$_G$ di visualizzazione non sia in grado di reperire o non abbia dati da mostrare all'utente per un determinato *pannello*$_G$.	Soddisfatto
ROF51	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi alla temperatura, espressa in gradi Celsius, il timestamp di rilevazione e le proprie coordinate geografiche.	Soddisfatto
ROF52	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi all'umidità, espressa in percentuale, il timestamp di rilevazione e le proprie coordinate geografiche.	Soddisfatto
ROF53	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi alla velocità del vento, espressa in chilometri all'ora (km/h), alla di-	Soddisfatto

	rezione del vento, espressa in gradi (con gli 0° a Est e i 180° a Ovest), il timestamp di rilevazione e le proprie coordinate geografiche.	
ROF54	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi alle precipitazioni, espresse in millimetri all'ora (mm/h), il timestamp di rilevazione e le proprie coordinate geografiche.	Soddisfatto
ROF55	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi all'inquinamento dell'aria, espresso in microgrammi al metro cubo (*PM10*$_G$), il timestamp di rilevazione e le proprie coordinate geografiche.	Soddisfatto
ROF56	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi alla percentuale di riempimento del bacino idrico controllato, il timestamp di rilevazione e le proprie coordinate geografiche.	Soddisfatto
ROF57	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi al numero di auto presenti all'interno del parcheggio controllato e il numero di posti auto totali a disposizione, il timestamp di rilevazione e le proprie coordinate geografiche.	Soddisfatto
ROF58	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi alla disponibilità della colonna di ricarica controllata, alla quantità di energia erogata dalla colonna, espresse in chilowatt all'ora (kWh), il timestamp di rilevazione e le proprie coordinate geografiche.	Soddisfatto
ROF59	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi alle coordinate geografiche della bicicletta elettrica controllata, la percentuale di batteria della stessa e il timestamp di rilevazione.	Soddisfatto
ROF60	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi alla percentuale di riempimento della zona ecologica controllata, il timestamp di rilevazione e le proprie coordinate geografiche.	Soddisfatto
ROF61	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi al numero di auto circolanti nella strada controllata, allo stato di congestione nella strada, espresso in stati (in ordine di congestione crescente sono: "LOW", "MEDIUM", "HIGH", "BLOCKED"), il tempo medio necessario per percorrere la strada, il timestamp di rilevazione e le proprie coordinate geografiche.	Soddisfatto
ROF62	Dev'essere realizzato un simulatore per almeno una tipologia di *sensore*$_G$.	Soddisfatto
ROF63	La simulazione deve produrre dati realistici, ovvero deve emulare il comportamento reale dell'entità che viene simulata.	Soddisfatto
RPF64	Il sistema deve rendere possibile la rilevazione di relazioni tra dati provenienti da sorgenti diverse.	Non soddisfatto
RPF65	Il sistema deve rendere possibile la previsione di eventi futuri, basata su dati storici e attuali.	Non soddisfatto

Table 5: Tabella dei requisiti funzionali con annesso stato di soddisfacimento.

Codice	Descrizione	Stato
ROV1	I dati vanno raccolti in un database OLAP, per esempio ClickHouse.	Soddisfatto
ROV2	I dati devono poter essere visualizzati su una piattaforma di data-visualization, per esempio Grafana.	Soddisfatto
ROV3	Deve essere utilizzato *Docker Compose*$_G$ versione 3.8 per l'installazione del software.	Soddisfatto
ROV4	I dati in ingresso nel database OLAP devono avere formato pseudo-tabellare, si deve utilizzare il formato Json.	Soddisfatto
ROV5	Deve essere utilizzato un message broker per lo streaming dei dati, per esempio Apache Kafka.	Soddisfatto
ROV6	Il sistema deve essere compatibile con Google Chrome v122, Mozilla Firefox v123 o Microsoft Edge v122.	Soddisfatto
ROV7	Il sistema deve poter essere installato su sistema operativo Windows (10 o 11), con RAM minimale di 6GB, processore 64 bit e compatibilità con WSL 2.	Soddisfatto
ROV8	Il sistema deve poter essere installato su sistema operativo MACOS (versione minima 10.14 Mojave) con RAM minimale di 6GB.	Soddisfatto
ROV9	Il sistema deve poter essere installato su sistema operativo Linux Ubuntu (22.04 o superiori) con RAM minimale di 6GB.	Soddisfatto

Table 6: Tabella dei requisiti di vincolo con annesso stato di soddisfacimento.

Codice	Descrizione	Stato
ROQ1	Il superamento di test che dimostrino il corretto funzionamento dei servizi utilizzati e delle funzionalità implementate. La copertura di test deve essere almeno dell'80% e deve essere dimostrata tramite report.	Soddisfatto
ROQ2	Viene richiesta una *documentazione*$_G$ sulle scelte implementative e progettuali, che dovranno essere accompagnate da motivazioni.	Soddisfatto
ROQ3	Viene richiesto il *Manuale Utente*.	Soddisfatto
ROQ4	Viene richiesto il documento *Specifica Tecnica*.	Soddisfatto
RDQ5	La documentazione dovrà riguardare anche problemi aperti ed eventuali possibili soluzioni da approfondire in futuro.	Non soddisfatto
RDQ6	L'*amministratore pubblico*$_G$ deve poter imparare a padroneggiare il *sistema*$_G$ in breve tempo.	Soddisfatto
ROQ7	La *repository*$_G$ di github del codice sorgente "InnovaCity" deve essere accessibile a tutti.	Soddisfatto
ROQ8	Devono essere rispettati i vincoli e le metriche definite nel *Piano di Qualifica v2.0*.	Soddisfatto

Table 7: Tabella dei requisiti di qualità con annesso stato di soddisfacimento.

4.2 Grafici requisiti soddisfatti

Riguardo alla soddisfazione dei requisiti il gruppo SWAT Engineering ha soddisfatto 77 su 82, arrivando ad una copertura del 94%.

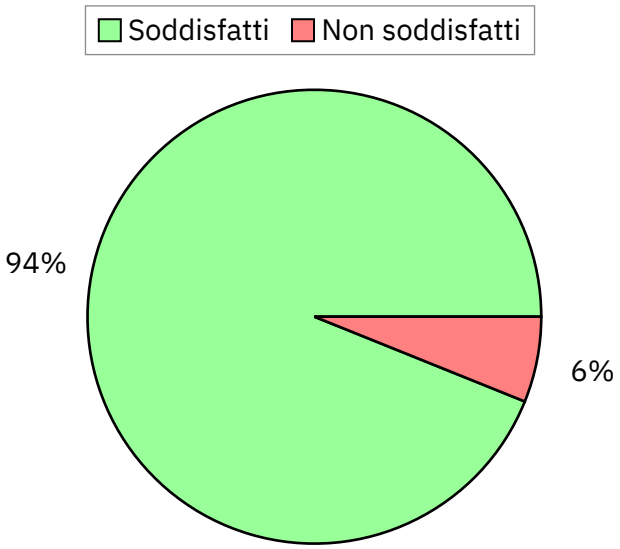

Grafico 1: Requisiti soddisfatti rispetto al totale.

Per quanto riguarda la copertura dei requisiti obbligatori, la copertura rilevata è di 56 su 56 requisiti, arrivando quindi ad un 100% sul totale.

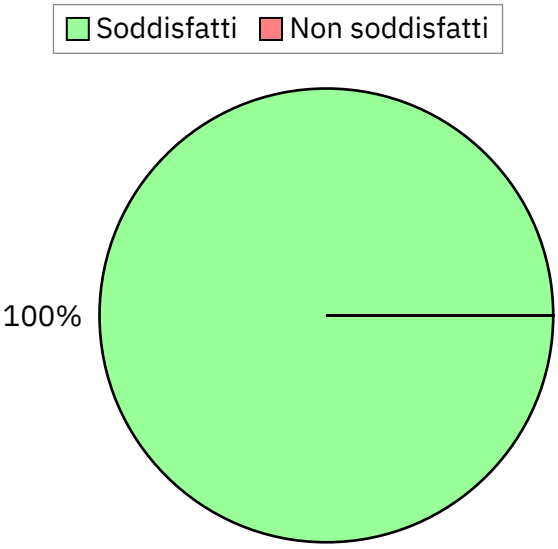

Grafico 2: Requisiti obbligatori soddisfatti rispetto al totale.

In termini di soddisfacimento dei requisiti desiderabili, è stata raggiunta una copertura del 91%, con 21 su 23.

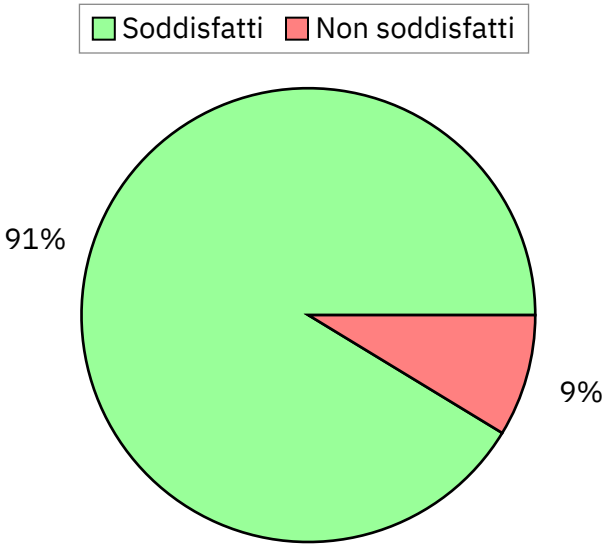

Grafico 3: Requisiti desiderabili soddisfatti rispetto al totale.

Per quanto concerne l'adempimento dei requisiti opzionali, abbiamo conseguito una percentuale del 33% sul totale, con 1 su 3 requisiti considerati.

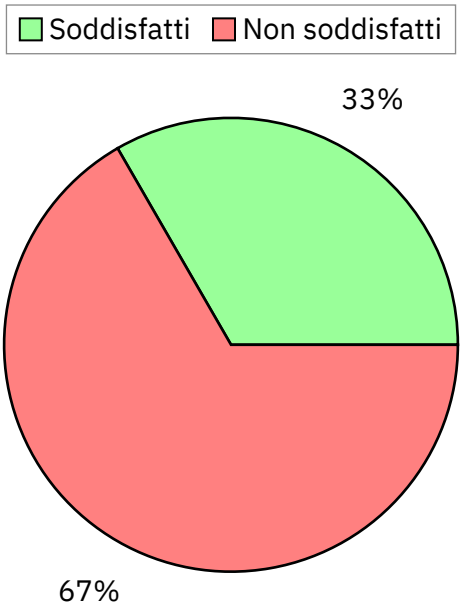

Grafico 4: Requisiti opzionali soddisfatti rispetto al totale.