S.W.A.T.
ENGINEERING

Specifiche Tecniche

Contatti: swateng.team@gmail.com

Versione: 0.2

Registro delle Modifiche

Versione	Data	Descrizione	Autore	Ruolo	Verificatore
0.2	4-03-2024	Stesura sezioni: Tecnologie, Architettura del sistema	Riccardo Toniolo, Giacomo D'Ovidio, Riccardo Alberto Costantin	Progettista	
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

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 "InnovaCity". La sua finalità primaria è fornire una descrizione dettagliata e approfondita di due aspetti centrali: l'architettura implementativa e l'architettura di deployment. Nel contesto dell'architettura implementativa, 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 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, garantire una completa 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 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 v1.0*, 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 (20/02/2024)
https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C6p.pdf (20/02/2024)

- *Norme di Progetto v1.0*
- *Analisi dei Requisiti v1.0*
- 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 (4/03/2024)
- 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/ (20/02/2024)
- **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 Codifica

2.1.1 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 ClickHouseDB.	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.	1.2.2
JSON	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 e formato dei messaggi spediti dai simulatori/sensori al broker dati.	2020-12

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

2.1.2 *Framework*$_G$ e librerie

Tecnologia	Descrizione	Versione
Pydantic	Libreria *Python*$_G$ per la serializzazione degli oggetti e per la validazione dei dati, inoltre si caratterizza dalla capacità di forzare i tipi garantendo che i dati siano coerenti durante l'elaborazione.	2.6
Confluent Kafka	Libreria *Python*$_G$ che fornisce gli strumenti necessari per produrre e consumare messaggi da Apache *Kafka*$_G$.	1.9

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

2.1.3 Database e servizi

Tecnologia	Descrizione	Versione
Clickhouse$_G$	Sistema di gestione dei database 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.	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*$_G$.	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.	10.3
Grafana$_G$ *ClickHouse*$_G$ Data Source	Plugin per *Grafana*$_G$ che consente di interrogare e visualizzare i dati di *ClickHouse*$_G$ in *Grafana*$_G$	4.0.3
Docker	Piattaforma open-source che permette di creare, distribuire e gestire applicazioni in contenitori virtuali.	25.0.3
Docker Compose	Strumento per definire e gestire applicazioni multi-container Docker attraverso file YAML.	2.24.6

Table 3: Tabella tecnologie per la Codifica: Database e servizi.

2.2 Per l'analisi del codice

Tecnologia	Descrizione	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.	1.7.1

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

3 Architettura del sistema

3.1 Architettura di implementazione

Il sistema 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 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;
- **κ-architecture**: semplifica la struttura della λ-architecture, eliminando il flusso batch. 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 sitema.

Per quanto appena descritto, la κ-architecture è la soluzione più adatta, in quanto specifica per il nostro caso d'uso. 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 ci porterebbe ad una complessità di manutenzione maggiore, logica di computazione duplicata e complessità di gestione del sistema in generale.

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

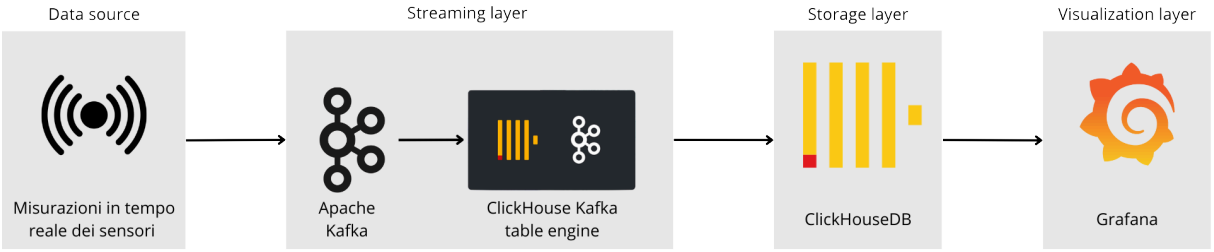

Figure 1: Schema delle componenti del sistema.

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

3.2 Diagramma del flusso di dati (data-flow)

Per illustrare il funzionamento del sistema, 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 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.

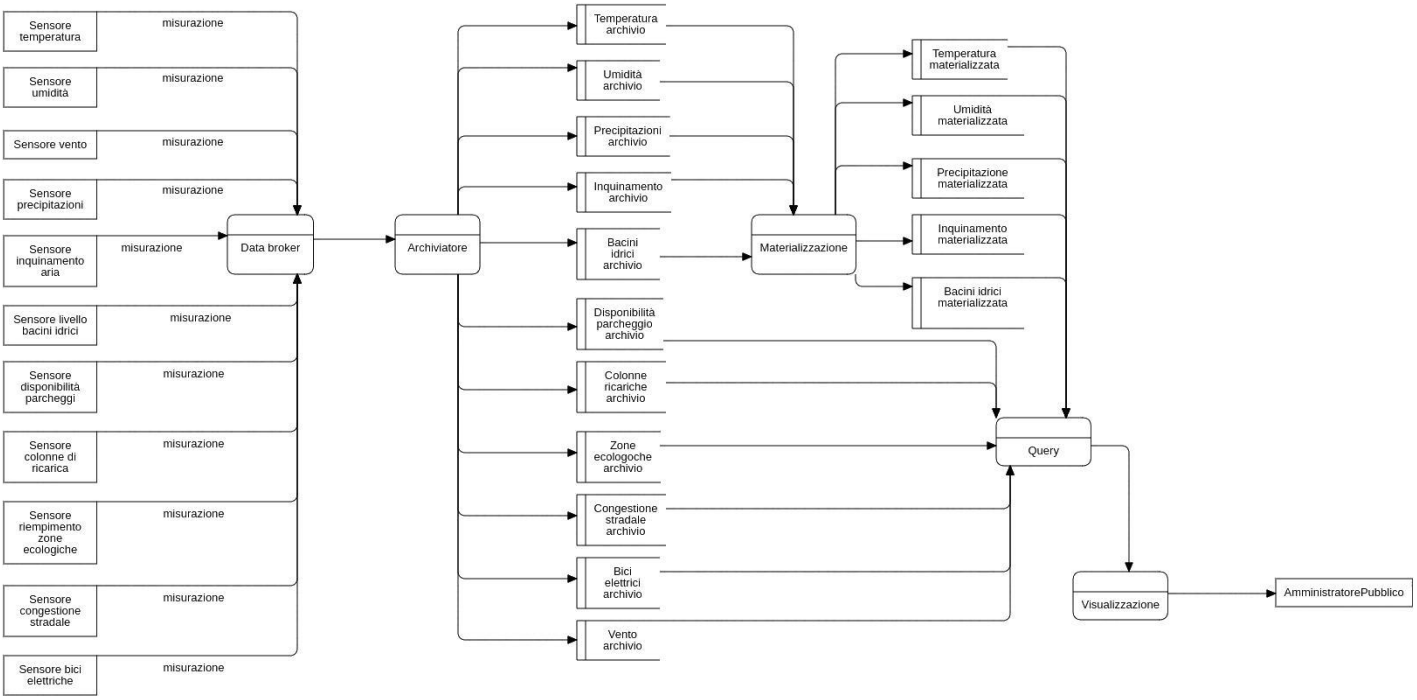

Figure 2: Diagramma data-flow relativo al percorso dei dati.

- **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. Wattaggio 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*$_G$. *Kafka*$_G$ 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 dei dati (*Kafka*$_G$). Questo avviene tramite la connessione a specifici topic nel broker dati, ognuno associato a un tipo di sensore distinto. Successivamente, i dati corrispondenti vengono archiviati nelle rispettive tabelle del database;

- **Materializzazione**: i dati corrispondenti la temperatura, umidità, precipitazioni, inquinamento atmosferico e livello dei bacini idrici vengono aggregati in tabelle apposite attraverso l'utilizzo di *materialized views*$_G$. Queste aggregazioni avvengono su intervalli di 5 minuti, consentendo così l'applicazione delle medie mobili;
- **Interrogazioni (query)**: vengono effettuate varie interrogazioni e analisi sui dati memorizzati all'interno delle tabelle;
- **Visualizzazione**: l'*amministratore pubblico*$_G$ visualizza i dati, ritornati in output dalle query, su una piattaforma apposita (nel nostro caso Grafana).

3.3 Architettura 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 individuata, tramite l'utilizzo di Diagrammi delle Classi e relative rapide descrizioni. Inoltre verranno motivati i design pattern individuati e le decisione progettuali rilevanti. Successivamente per ogni classe verranno illustrati metodi e attributi.

3.3.1 Struttura generale

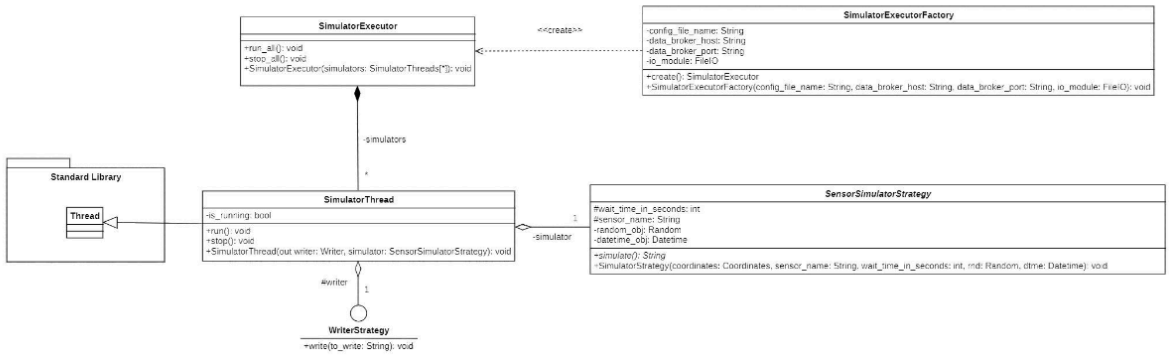

Figure 3: Diagramma delle classi 2

La classe *SimulatorExecutorFactory* è implementazione del design pattern *Factory*, si occupa prima della costruzione dei singoli simulatori a partire da un file di configurazione che gli viene passato tramite la costruzione, per poi restituire un SimulatorExecutor che avrà il compito di orchestrarli. 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.

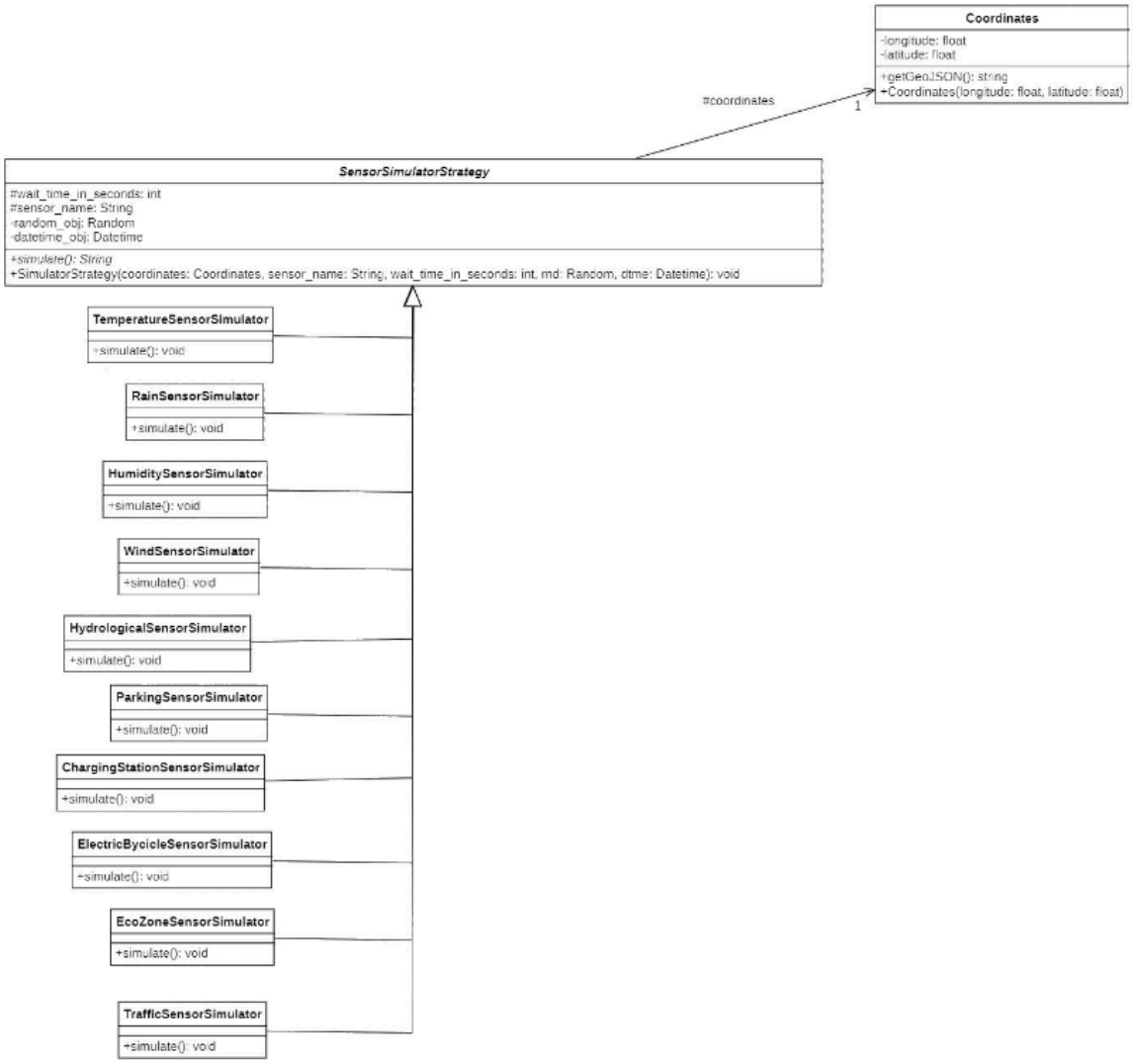

Figure 4: Diagramma delle classi 2

La classe *SensorSimulatorStrategy* è realizzazione del design pattern *Strategy*, dove ogni strategia rappresenta una tipologia di sensore 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*.

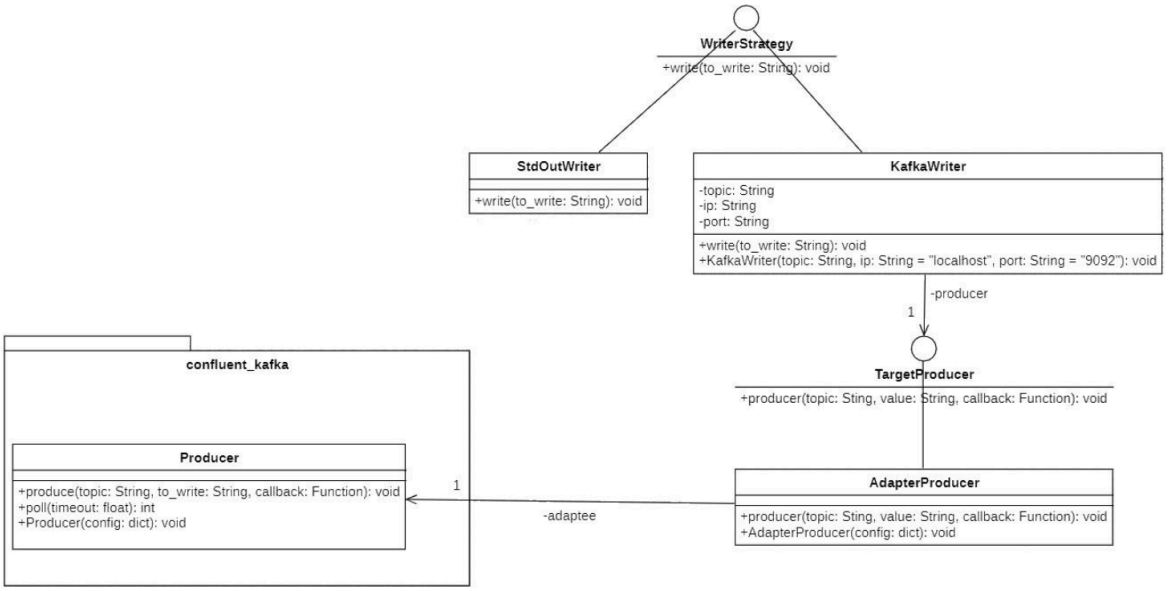

Figure 5: Diagramma delle classi 3

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 del Design Pattern potrebbe consentire di realizzare il componente di scrittura anche per eventuali ulteriori broker dati, nel momento in cui ce ne sia il bisogno. Nello specifico la classe *KafkaWriter* realizza il suo scopo tramite l'impiego del design pattern *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, spostando la complessità di tali proprio nell'adapter.

3.3.2 Classi: metodi e attributi

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

3.3.2.1 SimulatorExecutorFactory (Classe)

3.3.2.1.1 Attributi

- **config_file_name: String [Privato]**: Percorso relativo del file di configurazione dei simulatori;
- **data_broker_host: String [Privato]**: indirizzo ip della macchina che ospita il message broker;
- **data_broker_port: String [Privato]**: porta della macchina che ospita il message broker;
- **io_module: FileIO [Privato]**: modulo per interagire con il file system.

3.3.2.1.2 Metodi

- **create(): SimulatorExecutor [Pubblico]**: a partire dal file che viene passato tramite costruttore si occupa di costruire un oggetto della classe *SimulatorExecutor*.

3.3.2.2 SimulatorExecutor (Classe)

3.3.2.2.1 Attributi
- **simulators:SimulatorThread[*] [Privato]**: Collezione di oggetti *SimulatorThread*.

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

3.3.2.3 SimulatorThread (Classe)

3.3.2.3.1 Attributi
- **is_running: bool [Privato]**: Attributo che indica se al momento il simulatore sta producendo dati.
- **Se**

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

3.3.2.4 SensorSimulatorStrategy (Classe)

3.3.2.4.1 Attributi
- **wait_time_in_seconds: int [Protetto]**: intervallo temporale tra due simulazioni.
- **sensor_name: String [Protetto]**: nome identificativo del sensore;
- **random_obj: Random [Privato]**: oggetto della classe Random della libreria Standard di *Python*$_G$
- **datetime_obj: Datetime [Privato]**: oggetto della classe Datetime della libreria Standard di *Python*$_G$
- **coordinates : Coordinates [Protetto]**: oggetto della classe Coordinates.

3.3.2.4.2 Metodi
- **simulate(): String [Pubblico]**: metodo che produce una stringa contenente il messaggio contenente la rilevazione simulata effettuata.

3.3.2.5 Coordinates (Classe)

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

3.3.2.5.2 Metodi
- **getGeoJSON(): string [Pubblico]** : ritorna le coordinate geografiche nel formato GeoJSON.

3.3.2.6 WriterStrategy (interfaccia)

3.3.2.6.1 Metodi
- **write(in to_write:String): void [Pubblico]**: pubblica il messaggio contenuto nella stringa passata come parametro.

3.3.2.7 StdOutWriter (Classe)

3.3.2.7.1 Metodi

- **write(in to_write:String): void [Pubblico]**: Scrive il messaggio contenuto nella stringa passata come parametro tramite standard output.

3.3.2.8 KafkaWriter (Classe)

3.3.2.8.1 Attributi
- **ip: String [Privato]** : indirizzo ip della macchina che ospita il message broker;
- **port: String [Privato]** : porta della macchina che ospita il message broker;
- **topic: String [Privato]**: indica il topic sul message Broker in cui il messaggio sarà scritto.

3.3.2.8.2 Metodi
- **write(in to_write:String): void [Pubblico]**: Scrive il messaggio contenuto nella stringa passata come parametro sul topic corrispondente nel broker.

3.3.2.9 TargetProducer (Interfaccia)

3.3.2.9.1 Metodi
- **producer(in topic:Sting, in value:String, in callback:Function): void [Pubblico]**: Metodo che il client usa per inviare il messaggio a Kafka.

3.3.2.10 AdapterProducer (Interfaccia)

3.3.2.10.1 Attributi
- **adaptee: Producer (Privato)**

3.3.2.10.2 Metodi
- **producer(in topic:Sting, in value:String, in callback:Function): void [Pubblico]**: Metodo che richiama il metodo *produce()* dell'oggetto *adaptee* di tipo *Producer* della libreria *Confluent Kafka*.

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 funzionali 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:

- ROF - Requisito Obbligatorio Funzionale;
- RDF - Requisito Desiderabile Funzionale;
- RPF - Requisito Opzionale Funzionale.

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 Sensori, Ambientale, Urbanistica e Dati anomali & superamento soglie.	Soddisfatto
ROF3	L'utente deve poter visualizzare una *dashboard*$_G$ generale relativa ai sensori.	Non soddisfatto
ROF4	L'utente deve poter visualizzare le posizioni dei sensori come icone su una mappa, appartenente alla *dashboard*$_G$ generale relativa ai sensori.	Non soddisfatto
ROF5	L'utente deve poter visualizzare, in forma tabellare, l'elenco dei sensori con la relativa percentuale di batteria, un valore booleano che indica se il *sensore*$_G$ va a batteria autonoma o meno e la data di ultima manutenzione effettuata su tale *sensore*$_G$, all'interno della *dashboard*$_G$ generale relativa ai sensori.	Non soddisfatto
ROF6	L'utente deve poter monitorare i dati provenienti dai sensori relativi ai dati ambientali in una *dashboard*$_G$ apposita.	Non soddisfatto
ROF7	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante la media aritmetica della temperatura, espressa in gradi Celsius, per ciascun *sensore*$_G$, aggregando i dati per intervalli di 5 minuti, nella *dashboard*$_G$ relativa ai dati ambientali.	Non soddisfatto
ROF8	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante la media aritmetica della percentuale d'umidità, per ciascun *sensore*$_G$, aggregando i dati per intervalli di 5 minuti, nella *dashboard*$_G$ relativa ai dati ambientali.	Non soddisfatto
RDF9	L'utente deve poter visualizzare un *pannello*$_G$ contenente una mappa che evidenzi la direzione del vento, mediante frecce	Non soddisfatto

	aventi origine nelle coordinate geografiche del *sensore*$_G$, nella *dashboard*$_G$ relativa ai dati ambientali.	
ROF10	L'utente deve poter visualizzare un *pannello*$_G$ contenente una tabella la quale riporta l'ultima velocità del vento, espressa in chilometri all'ora, per ciascun *sensore*$_G$, nella *dashboard*$_G$ relativa ai dati ambientali.	Non soddisfatto
ROF11	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante la media aritmetica dell'intensità delle precipitazioni, espresse in millimetri all'ora (mm/h), per ciascun *sensore*$_G$, aggregando i dati per intervalli di 5 minuti, nella *dashboard*$_G$ relativa ai dati ambientali.	Non soddisfatto
RDF12	L'utente deve poter visualizzare un *pannello*$_G$ contenente un indice numerico, che esprime l'intensità media delle precipitazioni, espressa in millimetri all'ora (mm/h), degli ultimi 5 minuti, facendo la media dei dati raccolti tra tutti i sensori, nella *dashboard*$_G$ relativa ai dati ambientali.	Non soddisfatto
ROF13	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante la 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 5 minuti, nella *dashboard*$_G$ relativa ai dati ambientali.	Non soddisfatto
RDF14	L'utente deve poter visualizzare un *pannello*$_G$ contenente un indice numerico, che esprime l'inquinamento dell'aria medio, espressa in $\mu g/m^3$ (*PM10*$_G$), degli ultimi 5 minuti, facendo la media dei dati raccolti tra tutti i sensori, nella *dashboard*$_G$ relativa ai dati ambientali.	Non soddisfatto
ROF15	L'utente deve poter visualizzare un *pannello*$_G$ contenente un grafico in formato *time series*$_G$ rappresentante la percentuale di riempimento dei bacini idrici, per ciascun *sensore*$_G$, aggregando i dati per intervalli di 5 minuti, nella *dashboard*$_G$ relativa ai dati ambientali.	Non soddisfatto
RDF16	L'utente deve poter visualizzare un *pannello*$_G$ contenente un indice numerico, che esprime la temperatura media, espressa in gradi Celsius, degli ultimi 5 minuti, facendo la media dei dati raccolti tra tutti i sensori, nella *dashboard*$_G$ relativa ai dati ambientali.	Non soddisfatto
RDF17	L'utente deve poter visualizzare un *pannello*$_G$ contenente un indice numerico, che esprime l'inquinamento dell'aria massimo, espresso in $\mu g/m^3$ (*PM10*$_G$), degli ultimi 5 minuti, tra i dati registrati da tutti i sensori, nella *dashboard*$_G$ relativa ai dati ambientali.	Non soddisfatto
ROF18	L'utente deve poter monitorare i dati provenienti dai sensori relativi ai dati urbanistici in una *dashboard*$_G$ apposita.	Non soddisfatto
ROF19	L'utente deve poter visualizzare un *pannello*$_G$ contenente una mappa che evidenzi il numero di posti liberi nei vari parcheggi,	Non soddisfatto

	mediante indicatori numerici posti nelle coordinate geografiche del *sensore*$_G$, nella *dashboard*$_G$ relativa ai dati urbanistici.	
ROF20	L'utente deve poter visualizzare un *pannello*$_G$ contenente indicatori booleani posti nelle coordinate geografiche dei sensori che indichino la disponibilità delle colonne di ricarica, nella *dashboard*$_G$ relativa ai dati urbanistici.	Non soddisfatto
RDF21	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.	Non soddisfatto
ROF22	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 controllano queste, nella *dashboard*$_G$ relativa ai dati urbanistici.	Non soddisfatto
ROF23	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.	Non soddisfatto
ROF24	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.	Non soddisfatto
RDF25	L'utente deve poter monitorare i *dati anomali*$_G$ e i dati superanti delle soglie, in una *dashboard*$_G$ apposita.	Non soddisfatto
RPF26	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 anomali*$_G$ e superanti le soglie.	Non soddisfatto
RDF27	L'utente deve poter visualizzare un *pannello*$_G$ contenente una tabella che mostri i dati relativi a temperatura, precipitazioni, inquinamento dell'aria, bacini idrici e zone ecologiche, i cui valori superano una soglia fissata, il *sensore*$_G$ che li ha rilevati e il timestamp del rilevamento, nella *dashboard*$_G$ relativa ai *dati anomali*$_G$ e superanti le soglie.	Non soddisfatto
RDF28	L'utente deve poter visualizzare delle notifiche riguardo ad un valore di un dato di tipo temperatura, superante una soglia di 40° Celsius (40°C).	Non soddisfatto
RDF29	L'utente deve poter visualizzare delle notifiche riguardo ad un valore di un dato di tipo precipitazioni, superante una soglia di 50 millimetri all'ora (50mm/h).	Non soddisfatto

RDF30	L'utente deve poter visualizzare delle notifiche riguardo ad un valore di un dato di tipo inquinamento dell'aria (*PM10*$_G$), superante una soglia di 80 microgrammi su metro cubo.	Non soddisfatto
RDF31	L'utente deve poter visualizzare delle notifiche riguardo ad un valore di un dato di tipo percentuale riempimento bacini idrici, superante una soglia corrispondente al 70% della capienza di tale bacino.	Non soddisfatto
RDF32	L'utente deve poter visualizzare delle notifiche riguardo ad un valore di un dato di tipo percentuale riempimento zone ecologiche, superante una soglia corrispondente all'80% della capienza di tale zona.	Non soddisfatto
ROF33	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.	Non soddisfatto
ROF34	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.	Non soddisfatto
ROF35	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.	Non soddisfatto
ROF36	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.	Non soddisfatto
RDF37	Nei *pannelli*$_G$ con tabelle, l'utente deve poter ordinare i dati in base ai campi relativi al nome del *sensore*$_G$, alla percentuale di batteria, alla data di manutenzione del *sensore*$_G$, alla misurazione e al timestamp di quest'ultima (ove presenti), sia in ordine crescente che decrescente.	Non soddisfatto
RDF38	L'utente deve poter modificare il layout della *dashboard*$_G$ visualizzata, agendo sullo spostamento dei *pannelli*$_G$.	Non soddisfatto
RDF39	L'utente deve poter modificare il layout della *dashboard*$_G$ visualizzata, agendo sul ridimensionamento dei *pannelli*$_G$.	Non soddisfatto
ROF40	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*$_G$.	Non soddisfatto
ROF41	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.	Non soddisfatto
ROF42	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.	Non soddisfatto
ROF43	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi alla velocità del vento, espressa in chilometri all'ora, alla direzione	Non soddisfatto

	del vento, espressa in gradi (con gli 0° a Nord e i 180° a Sud), il timestamp di rilevazione e le proprie coordinate geografiche.	
ROF44	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.	Non soddisfatto
ROF45	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.	Non soddisfatto
ROF46	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.	Non soddisfatto
ROF47	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi al numero di parcheggi disponibili all'interno del parcheggio auto controllato, il timestamp di rilevazione e le proprie coordinate geografiche.	Non soddisfatto
ROF48	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi alla quantità di energia erogata dalla colonna di ricarica controllata, espresse in chilowatt all'ora (kWh), il timestamp di rilevazione e le proprie coordinate geografiche.	Non soddisfatto
ROF49	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.	Non soddisfatto
ROF50	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.	Non soddisfatto
ROF51	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi allo stato della congestione stradale nella strada controllata, espresse in stati (in ordine di crescente congestione sono: "LOW", "MEDIUM", "HIGH", "BLOCKED"), il timestamp di rilevazione e le proprie coordinate geografiche.	Non soddisfatto
RDF52	Il *sensore*$_G$ deve poter mandare e far persistere dati relativi al proprio stato, ovvero la propria percentuale di batteria (costantemente a 100% nel caso di sensori senza batteria autonoma), la data di ultima manutenzione effettuata su di esso, e la propria frequenza di inserimento dati espressa in secondi.	Non soddisfatto
ROF53	Dev'essere realizzato un simulatore per almeno una tipologia di *sensore*$_G$.	Non soddisfatto
ROF54	La simulazione deve produrre dati realistici.	Non soddisfatto
RPF55	Il sistema deve rendere possibile la rilevazione di relazioni tra dati provenienti da sorgenti diverse.	Non soddisfatto
RPF56	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.

4.2 Grafici requisiti soddisfatti

Riguardo alla soddisfazione dei vari requisiti funzionali, il gruppo SWAT Engineering ha soddisfatto 2 su 56 requisiti, arrivando ad una copertura del 4%.

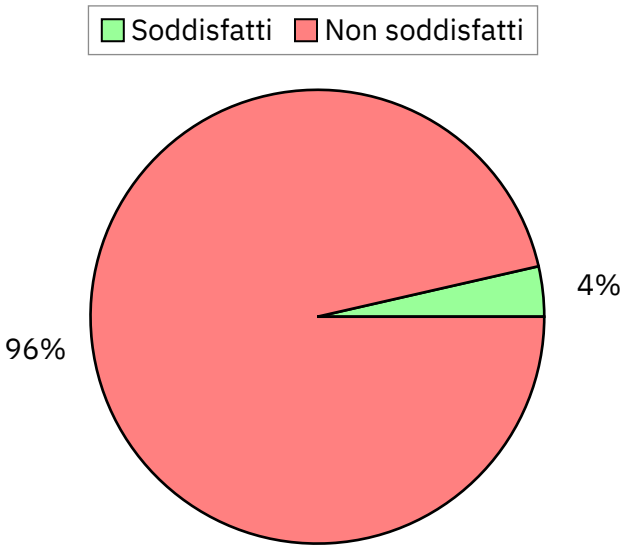

Grafico 1: Requisiti funzionali soddisfatti rispetto al totale.

Invece per quanto riguarda la copertura dei requisiti obbligatori, la copertura rilevata è di 6 su 54 requisiti, arrivando quindi ad un 11% sul totale.

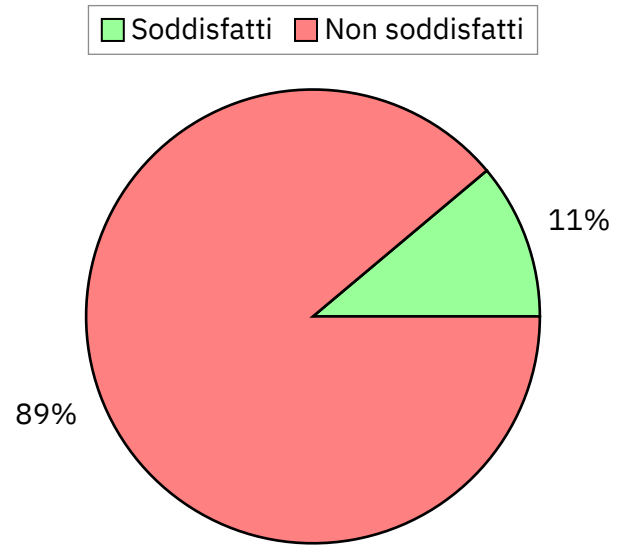

Grafico 2: Requisiti obbligatori soddisfatti rispetto al totale.