Laurea Triennale in Informatica - Università di Salerno

Corso di Ingegneria del Software - Proff. F. Ferrucci, F. Palomba

System Design Document

WasteGone

Riferimento	C01_SDD
Versione	1.0.0
Data	16/12/2024
Destinatario	Docenti di Ingegneria del Software 2024/25
Presentato da	C01 - Alessia Gatto (A.G.), Elisa Picilli (E.P.), Francesco Laudano (F.L.), Giovanni Croce (G.C.), Marco Iannuzzi (M.I.), Michela Palmieri (M.P.), Simon Carbone (S.C.)
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
08/11/2024	0.1.0	Prima stesura Proposed Architecture	G.C., S.C. e F.L.
09/11/2024	0.2.0	Introduzione	A.G. e M.I.
10/11/2024	0.2.1	Completamento Proposed Architecture	G.C., S.C. e F.L.
12/11/2024	0.2.2	Stesura sezione di Mapping H/S e Gestione Dati Persistenti	M.P. e E.P.
20/11/2024	0.2.3	Modifiche sezione di Mapping Hardware/Software	M.P. e E.P.
22/11/2024	0.2.4	Modifiche al Deployment Diagram	M.P. e E.P.
22/11/2024	0.3.0	Stesura sezione Boundary Condition	F.L. e M.P.
22/11/2024	0.3.1	Modifiche sezione Proposed architecture	G.C., S.C. e F.L
02/12/2024	0.3.2	Modifiche al Persistent Data Management	S.C.
02/12/2024	0.3.3	Modifiche al System decomposition	G.C.
16/12/2024	1.0.0	Revisione finale e consegna	Tutto il team

Indice

System Design Document (SDD) del Progetto WasteGone

1. Introduction

1.1 Purpose of the system

Il sistema è progettato per offrire un'app mobile che faciliti la gestione dei rifiuti domestici nelle aree urbane, promuovendo comportamenti ecologici e responsabilità ambientale tra i cittadini. L'app permetterà agli utenti di accedere a informazioni aggiornate sugli orari e sui punti di raccolta, di ricevere notifiche per il corretto conferimento e di consultare le categorie di rifiuti per una raccolta differenziata più precisa.

Il sistema implementerà funzionalità avanzate per semplificare ulteriormente le operazioni, inclusa la possibilità di segnalare discariche abusive, l'accesso ad una mappa interattiva che indicherà i punti di raccolta di rifiuti speciali, industriali o pericolosi.

Questo sistema non solo migliorerà l'efficienza della raccolta dei rifiuti, ma contribuirà anche a ridurre gli errori di conferimento, con l'obiettivo di diminuire l'impatto ambientale dei rifiuti urbani. Sarà, inoltre, un canale di educazione ambientale, in quanto fornirà eventi e iniziative locali che sensibilizzeranno i cittadini verso pratiche sostenibili e innovative nella gestione dei rifiuti.

1.2 Design Goals

In questa sezione verranno presentati i Design Goals, ovvero le qualità del sistema che dovranno essere migliorate e ottimizzate. I Design Goals scelti sono divisi nelle seguenti categorie:

- **Dependability**
- **Performance**
- **Maintenance**
- **End user**

Ogni design goal è descritto da:

- **Rank**: Un livello di priorità compreso da 1 a 10
- **ID Design Goal**: Identificativo univoco
- **Descrizione**: Descrizione del design goal
- **Categoria**: Categoria di appartenenza del design goal
- **RNF di origine**: Requisito non funzionale da cui deriva

Rank	ID	Nome	Descrizione	Categoria	RNF di origine
1	DG_01	Disponibilità 24/7	Il sistema deve essere disponibile 24/7.	Dependability	RNF_01
2	DG_02	Gestione malfunzionamenti	Il sistema deve gestire automaticamente eventuali errori o malfunzionamenti, con un ripristino completo delle funzionalità entro un intervallo di tempo prestabilito, per garantire la continuità del servizio.	Dependability	RNF_03
3	DG_03	Scalabilità	Il sistema deve supportare l'aggiunta dinamica di server o risorse senza interrompere il servizio per gestire aumenti di carico.	Performance	RNF_05
4	DG_04	Protezione dei dati	Il sistema deve rispettare le normative sulla protezione dei dati, garantendo che gli utenti possano accedere, rettificare e cancellare i propri dati su richiesta.	Dependability	RNF_04
5	DG_05	Pagamento sicuro	Il sistema deve offrire un pagamento sicuro e tracciato per i	Dependability	RNF_02

			pagamenti delle tasse sui rifiuti.		
6	DG_06	Documentazione degli artefatti	Il sistema deve garantire che il codice e l'infrastruttura debbano essere ben documentati per facilitare la manutenzione e l'aggiornamento da parte di tecnici diversi, riducendo il rischio di errori futuri.	Maintenance	RNF_07
7	DG_07	Sistema di notificazione	Il sistema deve inviare agli utenti le notifiche giornaliere riguardanti i rifiuti da conferire entro un minuto dall'orario programmato.	Performance	RNF_06
8	DG_08	Evoluzione	Il sistema deve poter essere facilmente evoluto nel tempo, garantendo la realizzazione delle funzionalità mancanti, entro un numero prestabilito di nuove release.	Maintenance	RNF_10
9	DG_09	Backup	Il sistema deve effettuare backup regolari dei dati.	Dependability	RNF_09
10	DG_10	Sezione help	Il sistema deve prevedere una sezione help dove poter consultare un manuale utente e segnalare malfunzionamenti.	End user	RNF_12

1.2.1 Trade-offs

Trade-Off	Descrizione
Performance vs Dependability	Per garantire un'elevata disponibilità del sistema, potrebbe essere sacrificata la velocità con cui il sistema recupera le funzionalità dopo un guasto. Un sistema sempre attivo richiede risorse costanti e monitoraggio, che possono ritardare i processi di recupero immediato.
Performance vs End User	Avere una sezione di supporto e assistenza è importante per migliorare l'esperienza utente, ma può ridurre le prestazioni del sistema, specialmente durante i picchi di accesso. Risorse come documentazione e gestione delle richieste di supporto possono rallentare il sistema, soprattutto se le richieste sono elevate.
End User vs Dependability	Per aumentare la sicurezza dei dati, l'usabilità del sistema può essere compromessa. Ad esempio, l'uso di password complesse, autenticazione a due fattori e controlli frequenti garantisce maggiore protezione, ma rende l'esperienza utente meno fluida e richiede più passaggi per l'accesso.
Performance vs Maintenance	L'ottimizzazione delle prestazioni spesso implica implementazioni specifiche e ottimizzazioni ad hoc, che possono rendere il sistema meno flessibile e più difficile da manutenere.

1.3 Definition, acronyms, e abbreviations

- **RNF**: NonFunctional Requirements
- **RAD**: Requirements Analysis Document
- **DG**: Design Goal
- **SOW**: Statement of Work
- **XML**: eXtensible Markup Language
- **API**: Application Programming Interface

1.4 References

Di seguito una lista di riferimenti ad altri documenti a cui si fa riferimento:

- **SOW**
- **RAD**
- **Slide del corso**
- **Libri:**
    - **Ian Sommerville - Software Engineering, 10th Edition-Pearson**
    - **Prentice Hall – Pearson – Object-Oriented Software Engineering – Using UML, Patterns and Java. Autori: Bernd Bruegge & Allen H. Dutoit**

1.5 Overview

Il presente documento di System Design è organizzato in tre sezioni principali:

1. **Introduzione**: Una panoramica generale che descrive lo scopo del sistema e gli obiettivi di design che si intende perseguire.
2. **Architettura software corrente**: La descrizione dello stato attuale dell'architettura di qualsiasi software esistente.
3. **Architettura software proposta**: Una descrizione dettagliata della suddivisione del sistema in sottosistemi, del mapping hardware/software e della gestione dei dati persistenti.

2. Current architecture

Le funzionalità offerte da WasteGone non trovano al momento un corrispettivo diretto in un unico sistema disponibile sul mercato.

I software attualmente esistenti integrano solo alcune delle sue caratteristiche, rendendo impossibile il confronto con un'architettura già consolidata.

3. Proposed architecture

3.1 Overview

Il sistema adotta un'architettura **Fat Client** (una variante dell'architettura Client-Server), composta da tre Layer software e due Tier hardware, particolarmente adatta per un'applicazione nativa Android che richiede operazioni rapide e una gestione locale della logica applicativa. Questo approccio concentra gran parte della logica e delle funzionalità sull'app client, minimizzando la dipendenza da un server centrale per l'elaborazione. Così facendo l'applicazione Android è in grado di elaborare la maggior parte delle operazioni direttamente sul dispositivo, migliorando l'efficienza e riducendo la necessità di connessioni frequenti con il server.

L'architettura è strutturata come segue:

- **Client Android (Presentation e Business):** L'applicazione è responsabile sia della logica di presentazione che della logica applicativa, integrando la gestione dei dati. XML sarà impiegato per definire l'interfaccia grafica, mentre Java si occuperà della logica di business, che verrà eseguita localmente per ridurre i tempi di risposta e migliorare l'usabilità.
- **Server (Persistence):** Firestore è utilizzato per l'archiviazione e il recupero dei dati necessari. Sebbene il client gestisca in modo autonomo gran parte della logica, l'accesso a Firestore garantisce una sincronizzazione affidabile per i dati condivisi tra utenti e dispositivi.

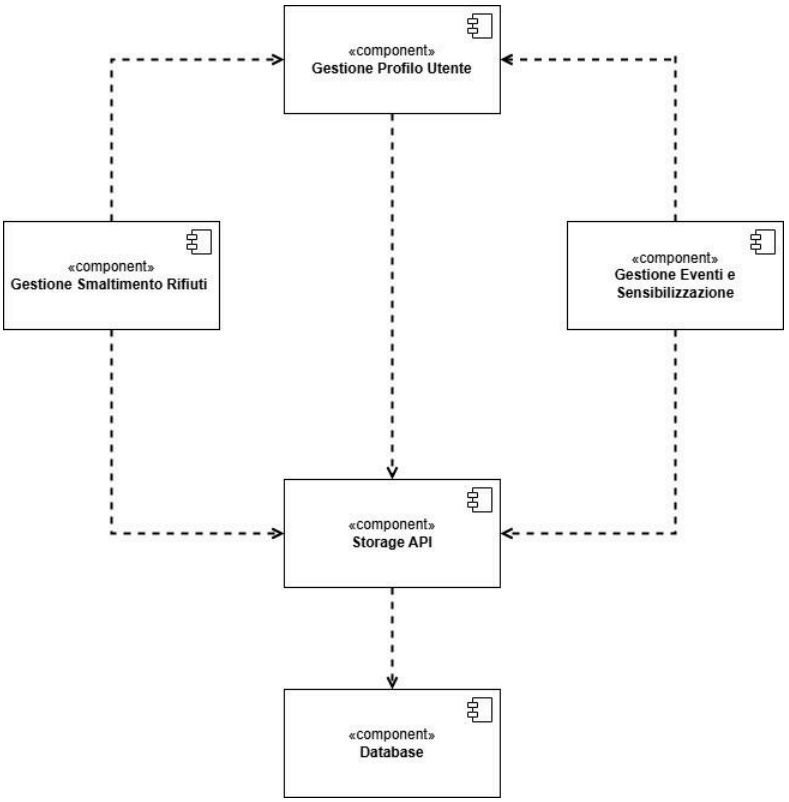
3.2 System decomposition

I sottosistemi individuati sono:

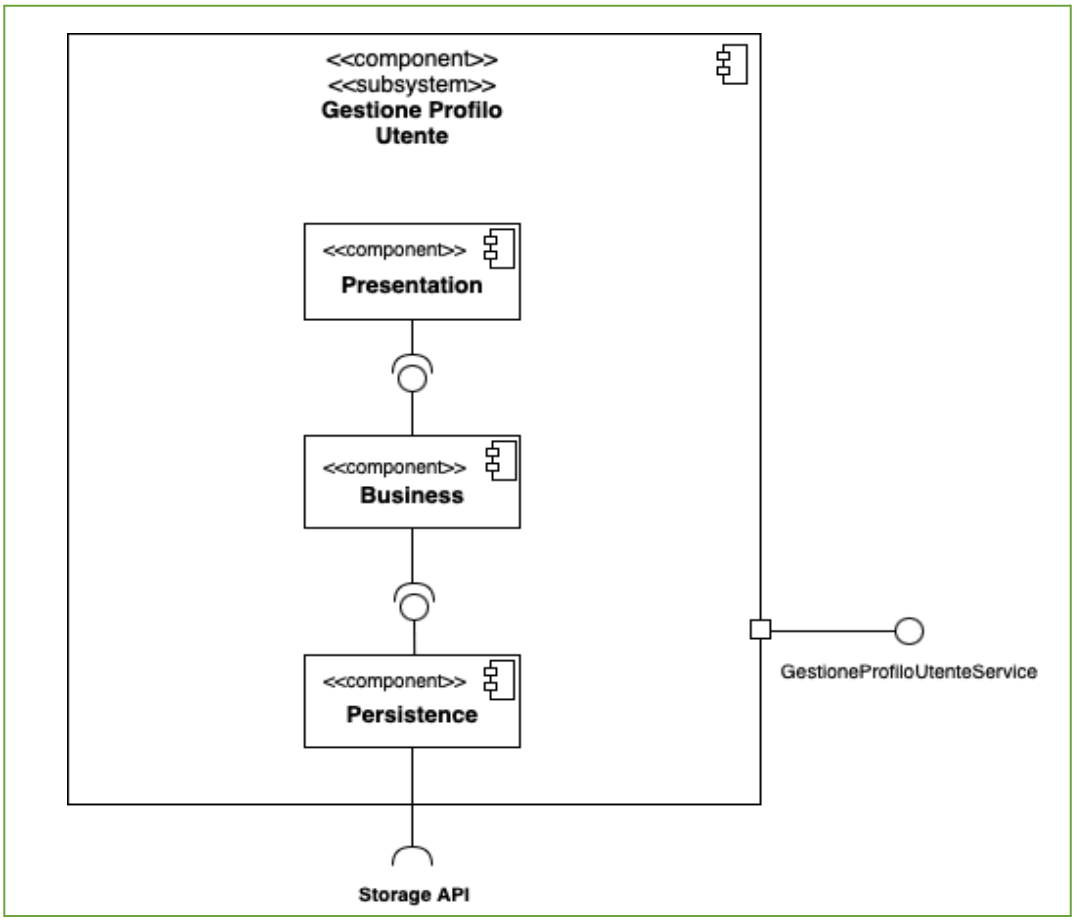
- **Gestione Profilo Utente:** Si occupa delle funzionalità di Login, Logout, visualizzazione area utente, con le relative notifiche e la modifica dati account.
- **Gestione Eventi e Sensibilizzazione:** Si occupa della visualizzazione dei vari eventi sul territorio e della visualizzazione delle tasse.
- **Gestione Smaltimento Rifiuti:** Si occupa della visualizzazione dei punti di ritiro e dei dettagli sui rifiuti.
- **Storage API**: Si interpone tra i vari sottosistemi e il sottosistema di Database.
- **Database:** Si occupa di gestire le operazioni di persistenza dei dati, con il Database associato.

Sono mostrate di seguito le dipendenze tra i sottosistemi attraverso un **Component Diagram**.

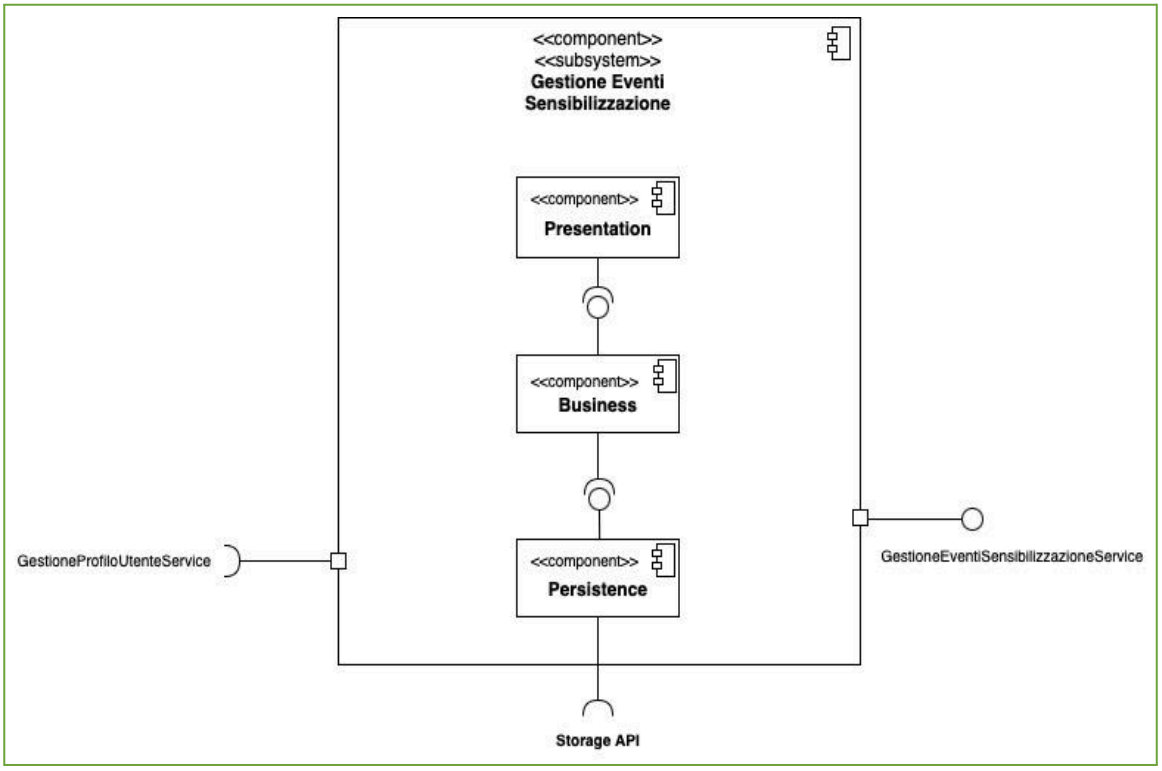
3.2.1 Component Diagram



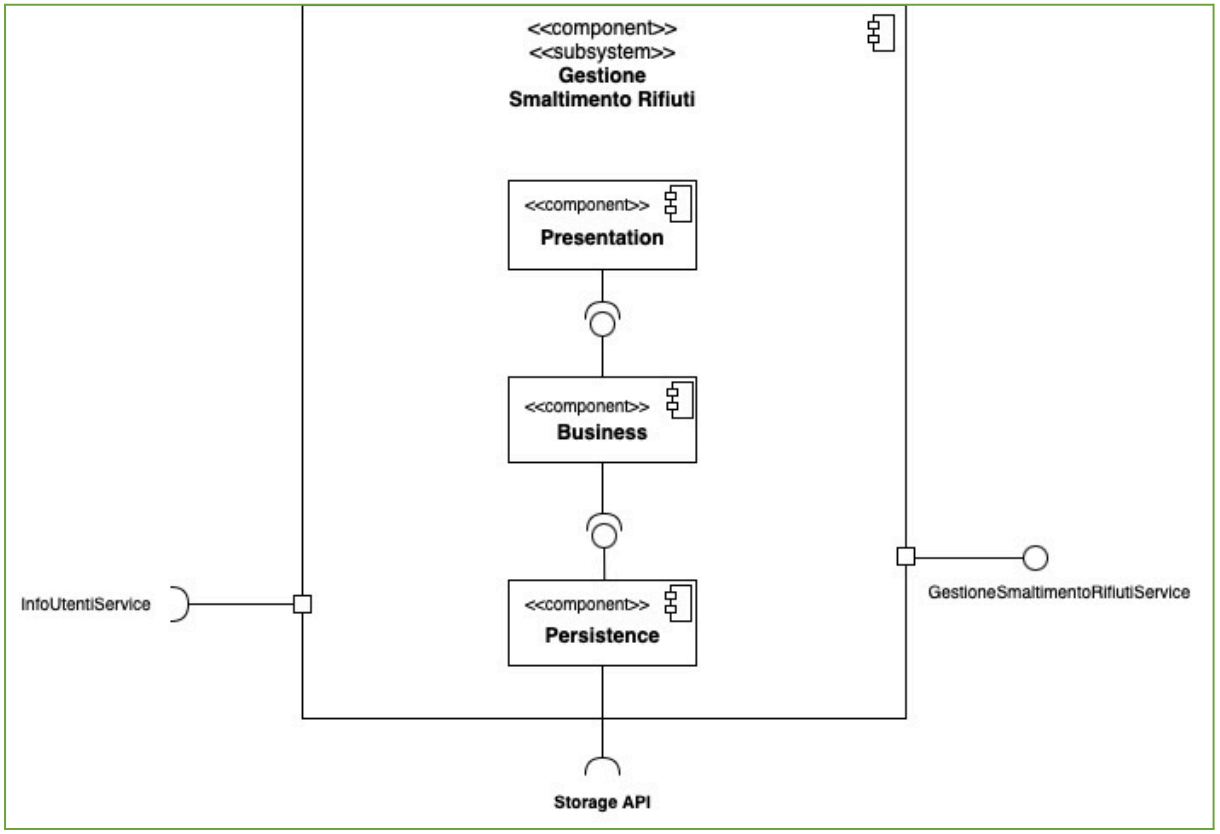
3.2.2 Sottosistema Gestione Profilo Utente



3.2.3 Sottosistema Gestione Eventi e Sensibilizzazione



3.2.4 Sottosistema Gestione Smaltimento Rifiuti



3.3 Mapping Hardware/Software

L'applicazione Android integrerà sia la logica di presentazione che quella di elaborazione direttamente sul dispositivo mobile. Il database Firestore, invece, sarà ospitato su un server fornito da Google Cloud Platform, che include un'istanza della suite di servizi Firebase. Le comunicazioni con il database avverranno tramite le API fornite dal servizio. Pertanto, il sistema sfrutta l'interazione tra due dispositivi distinti.

3.3.1 Interface and Application Logic Layer: Android Device

Formato dalle seguenti componenti:

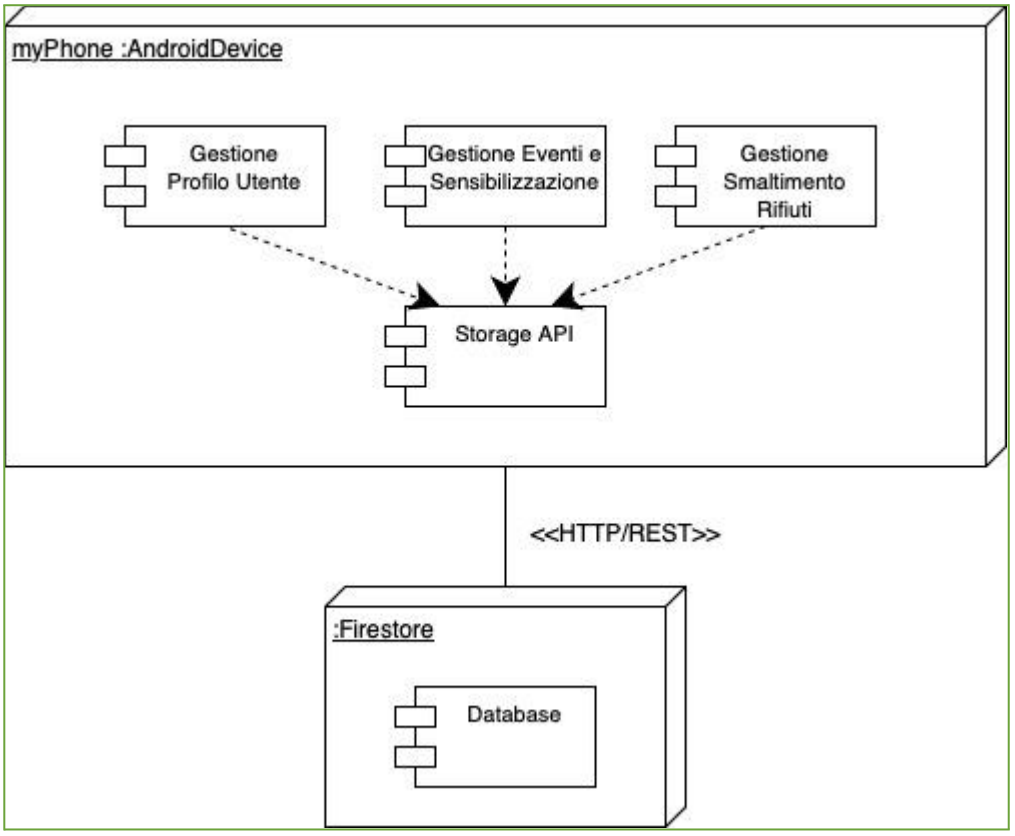
- ***Gestione Profilo Utente*** che gestisce il login, logout, le notifiche e la visualizzazione e modifica del profilo.
- ***Gestione Eventi e Sensibilizzazione*** che mostra eventi e informazioni sulle tasse.
- ***Gestione Smaltimento Rifiuti*** che fornisce i dettagli sul conferimento dei rifiuti.
- ***Storage API*** che rappresenta l'interfaccia che comunica con il database tramite un'interfaccia HTTP/REST.

3.3.2 Storage Layer: Firestore

Formato dalla componente Database che gestisce l'archiviazione dei dati per tutti i sottosistemi.
La scelta di questo modello architetturale è motivata dalla netta separazione tra la logica di presentazione e di business, da quella della gestione dei dati.
Inoltre garantisce:

- **Manutenibilità**;
- **Scalabilità**;
- **Riutilizzabilità**;
- **Sicurezza**;
- **Affidabilità.**

Di seguito è mostrato il **Deployment Diagram**.



3.4 Persistent Data Management

Per la gestione e il salvataggio dei dati persistenti è stato scelto il database Firestore, un database di tipo NoSQL orientato ai documenti. Questa scelta è stata motivata dai seguenti fattori:

1. **Sincronizzazione in tempo reale**: Firestore è progettato per supportare la sincronizzazione dei dati in tempo reale su tutti i dispositivi connessi. Ciò è utile per app che richiedono aggiornamenti immediati (es. notifiche).
2. **Supporto offline**: Firestore consente l'accesso ai dati anche in modalità offline. I dati vengono sincronizzati automaticamente quando la connessione torna disponibile, migliorando l'esperienza dell'utente e la disponibilità dell'app.
3. **Scalabilità automatica**: Firestore è un database serverless e gestisce automaticamente il ridimensionamento in base al carico di lavoro. Non è necessario gestire l'infrastruttura o preoccuparsi della capacità del server.
4. **Integrazione nativa con Firebase**: Firestore è parte dell'ecosistema Firebase, che offre strumenti per autenticazione, analytics, notifiche push e molto altro. L'integrazione semplifica la gestione di vari servizi essenziali in un'app Android. Inoltre grazie all'integrazione con Firebase, Firestore riduce i tempi di sviluppo e facilita l'implementazione.
5. **Sicurezza e controllo degli accessi**: Firestore offre un sistema avanzato di regole di sicurezza basate sul ruolo e sull'autenticazione Firebase. Questo permette di impostare facilmente permessi di lettura e scrittura specifici per utenti o gruppi.
6. **Affidabilità e manutenzione zero**: Essendo un servizio cloud gestito da Google, Firestore offre elevata affidabilità e non richiede manutenzione del server. Gli aggiornamenti e il backup sono gestiti automaticamente.

3.4.1 Class Diagram Ristrutturato

Si è deciso di accorpare la generalizzazione di RifiutoCasa e RifiutoSpeciale in un'unica entità Rifiuto (in modo da trattare solo i rifiuti che vengono conferiti alla porta) e lasciare l'entità "Prenotazione" per il ritiro di un rifiuto speciale alla porta. Questo approccio permette di ridurre la complessità delle relazioni. Inoltre, l'entità "Segnalazione" e "PuntoRitiro" sono state eliminate per semplificare la struttura del sistema e ridurne la complessità.
Infatti l'attuale implementazione del sistema prevede le seguenti entità: "Rifiuto", "Prenotazione", "Utente", "Evento" e "Tassa" per rispondere in modo preciso e completo alle esigenze del sistema di gestione dei rifiuti.
Assumiamo che:

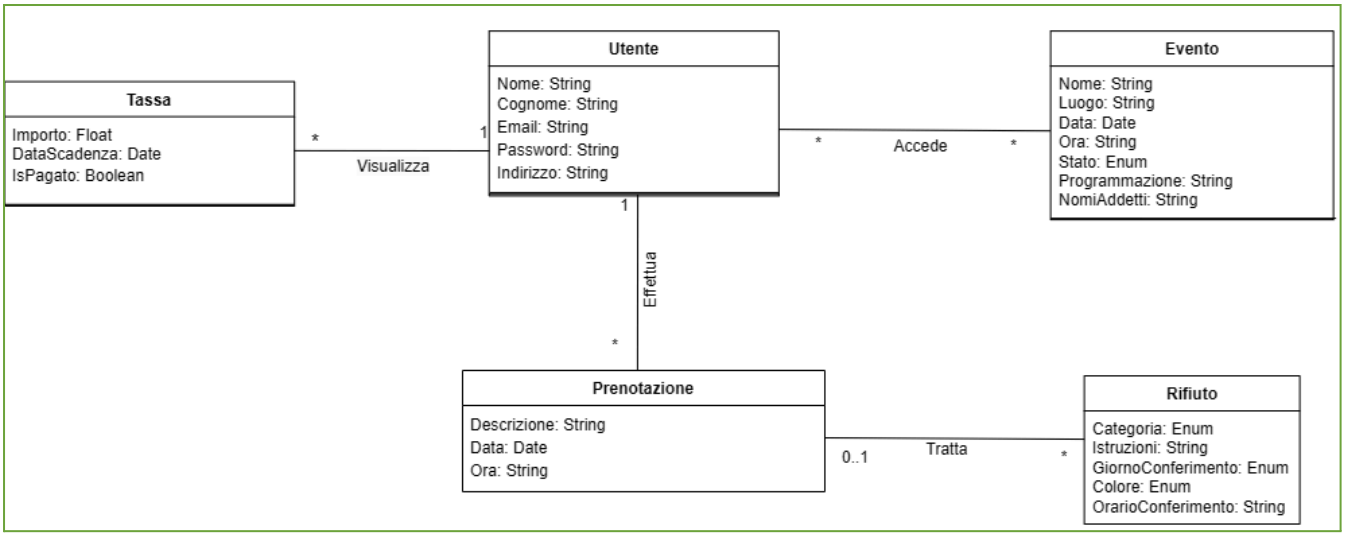
- In "Evento", "Stato" può assumere i seguenti valori: In corso, Sospeso, Terminato, In programma.

- In "Rifiuto", "GiornoConferimento" può assumere i seguenti valori: Lunedì, Martedì, Mercoledì, Giovedì, Venerdì, Sabato.

- In "Rifiuto", "Categoria" può assumere i seguenti valori: Plastica, Carta, Vetro, Alluminio, Indifferenziata, Umido.

- In "Rifiuto", "Colore" può assumere i seguenti valori: Rosso, Verde, Giallo, Blu, Viola, Arancione.

Di seguito è riportato il **Class Diagram** ristrutturato:



3.5 Global Software Control

Il sistema adotta un controllo centralizzato e globale del software, di tipo **event-driven**, in cui un componente principale supervisiona tutte le operazioni in base agli eventi che si verificano. Questi eventi sono generati dalle interazioni dell'utente con l'interfaccia grafica.

La scelta di un controllo di tipo event-driven consente al componente centrale di gestire gli eventi, permettendo agli altri moduli del sistema di focalizzarsi sulle loro funzioni specifiche. Inoltre, questo approccio centralizzato facilita la gestione degli errori, semplificando la risoluzione dei problemi e riducendo gli impatti sull'intero sistema.

3.6 Boundary Condition

3.6.1 Inizializzazione

Il sistema si avvia quando l'utente apre l'app Android, attivando la gestione del profilo utente, la visualizzazione degli eventi e la connessione al database Firestore. Il sistema si sincronizza con il server Firebase per garantire il caricamento dei dati necessari.

3.6.2 Fallimento

I fallimenti possono derivare da errori nell'autenticazione, problemi nella connessione a Firestore, o interruzioni nella rete. In caso di errori critici (e.g., disconnessione del database), l'app mostra messaggi di errore e tenta il ripristino automatico quando la connessione è disponibile.

3.6.3 Terminazione

Quando l'utente esce dall'app, tutte le risorse vengono rilasciate e i dati non ancora sincronizzati vengono memorizzati per il prossimo avvio. Eventuali notifiche in sospeso vengono inviate prima della disconnessione.