



UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE
DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

ELABORATO SOFTWARE ARCHITECTURE DESIGN

Pacifico Catapano M63001330

Fabio Boccia M63001541

Salvatore Moggio M63001534

Simone Dotolo M63001503

A.A. 2022-2023

Indice

1	Introduzione al progetto	1
1.1	Dati GPS	1
2	Analisi e specifica dei requisiti	3
2.1	Attori	3
2.2	Requisiti	4
2.2.1	Requisiti funzionali	4
2.2.2	Requisiti di sistema	4
2.2.3	Requisiti non funzionali	5
2.3	Diagramma di contesto	6
2.4	Casi d'uso	6
2.4.1	Descrizione dei casi d'uso	7
2.5	Stima dei costi	9
2.5.1	Unadjusted Use Case Weight (UUCW)	9
2.5.2	Unadjusted Actor Weight (UAW)	10
2.5.3	Technical Complexity Factor (TCF)	11
2.5.4	Environmental Complexity Factor (ECF)	12
2.5.5	Total Use Case Points (UCP)	12
2.6	System Domain Model	13
2.7	Diagrammi di sequenza	13
3	Progettazione	16
3.1	Scelte progettuali	16
3.2	Pattern Architetturale - Client-Server	16
3.2.1	Client	17
3.2.2	Server	17
3.3	Stile Architetturale - Service-oriented (SOA)	17
3.4	REST	18
3.4.1	Client - Server in REST	18
3.4.2	Stateless	19
3.4.3	Cacheable	19
3.4.4	Interfaccia uniforme	20
3.4.5	Sistema stratificato	20

3.5	Diagramma dei componenti	20
3.6	Diagramma delle classi	21
3.7	Diagrammi di sequenza di progettazione	23
3.8	Diagramma di deployment	26
4	Implementazione	27
4.1	Scelte di sviluppo	27
4.2	Framework di sviluppo	27
4.2.1	Back-end: FLASK	28
4.2.2	Gestione mail: SENDGRID	29
4.3	Pattern Proxy	30
4.4	Entity Relationship Diagram	31
5	Testing	32
5.1	Test funzionale	32
5.1.1	Test di unità	32
5.1.2	GitHub Actions	34
	Glossario	35

Introduzione al progetto

L'applicazione è stata progettata per rispondere alle esigenze degli utenti che lavorano con dati geospaziali, fornendo loro un ambiente dedicato per il salvataggio e la visualizzazione dei dati geospaziali. Questi dati provengono da sistemi di posizionamento globale (GPS) con precisione al centimetro. I dati geospaziali vengono prima ripuliti da un servizio esterno, chiamato U-BLOX, e dopo inviati ad un server.

Essa mira a fornire agli utenti uno strumento intuitivo e potente per sfruttare appieno il potenziale informativo dei dati geospaziali.

Il nostro sistema deve ricevere un file UBX proveniente da un computer esterno; si deve effettuare il parsing per ricavarne i dati geospaziali e salvarli nel database.

1.1 Dati GPS

I dati GPS possono presentarsi in due formati differenti:

1. Coordinate geodetiche
2. Coordinate ECEF

Coordinate geodetiche

Le coordinate geodetiche sono un sistema di coordinate utilizzato per descrivere la posizione di un punto sulla superficie terrestre in modo preciso. Questo sistema di coordinate tiene conto della forma ellissoidale della Terra e fornisce informazioni sulla latitudine, longitudine e altitudine di un punto specifico.

Nel sistema di coordinate geodetiche, la latitudine misura la distanza angolare di un punto rispetto all'equatore. È misurata in gradi, minuti e secondi o in forma decimale (ad esempio, 40.7128° N). La latitudine varia da -90° (corrispondente al polo sud) a $+90^\circ$ (corrispondente al polo nord).

La longitudine, invece, misura la distanza angolare di un punto rispetto al meridiano di riferimento, solitamente il meridiano di Greenwich. Anche la longitudine è espressa in gradi,

minuti e secondi o in forma decimale (ad esempio, 74.0060° W per New York). La longitudine varia da -180° a $+180^\circ$, coprendo l'intera circonferenza terrestre.

Infine, l'altitudine o elevazione indica la distanza verticale di un punto rispetto a un riferimento, solitamente il livello del mare. Può essere espressa in metri o piedi.

Le coordinate geodetiche forniscono un modo conveniente per specificare la posizione di un punto sulla Terra in modo univoco e sono ampiamente utilizzate in applicazioni come la navigazione, la cartografia, la geodesia e l'ingegneria geotecnica.

Coordinate ECEF

Le coordinate ECEF (Earth-Centered, Earth-Fixed) sono un sistema di coordinate utilizzato per descrivere la posizione di un punto sulla superficie terrestre o nello spazio in relazione al centro della Terra. Questo sistema di coordinate è basato su un modello ellissoide di riferimento della Terra, che approssima la forma del nostro pianeta.

Nelle coordinate ECEF, il sistema di riferimento è centrato nel centro di massa della Terra e l'asse Z è allineato con l'asse di rotazione terrestre (polo nord). L'asse X punta verso l'intersezione dell'equatore con il meridiano di Greenwich, e l'asse Y è ortogonale a entrambi gli assi X e Z, completando così un sistema di coordinate cartesiane a tre dimensioni.

Le coordinate ECEF sono espresse solitamente in metri. L'origine del sistema di coordinate ECEF corrisponde al centro geometrico della Terra. Quindi, la posizione di un punto sulla superficie terrestre o nello spazio può essere definita tramite le sue coordinate X, Y e Z rispetto al sistema ECEF.

Le coordinate ECEF sono ampiamente utilizzate in ambiti come la navigazione satellitare, la geodesia, l'astronomia e altre discipline che richiedono una precisa localizzazione sulla Terra.

Analisi e specifica dei requisiti

Il presente capitolo si focalizza sull'analisi delle specifiche e dei requisiti del software per la gestione dei dati geospaziali. Prima di elencare in dettaglio i requisiti necessari, è importante comprendere il contesto e gli obiettivi del sistema.

2.1 Attori

Gli attori che interagiscono con il sistema sono i seguenti:

- **Utente non autenticato**
- **Utente autenticato**
- **IoT device**

Utente non autenticato

L'utente non autenticato per visualizzare i dati deve accedere al sistema.

Utente autenticato

L'utente deve effettuare il login con le proprie credenziali per visualizzare i dati.

IoT device

Il dispositivo IoT invia i dati geospaziali che il nostro sistema deve ricevere e salvare in opportune directory.

2.2 Requisiti

I requisiti sono le specifiche delle funzionalità, delle prestazioni e degli attributi di qualità che un sistema software deve soddisfare per essere considerato accettabile e utile per gli utenti finali. I requisiti rappresentano le necessità, le aspettative e le restrizioni che guidano la progettazione, lo sviluppo e la valutazione del software.

2.2.1 Requisiti funzionali

Descrivono le funzionalità specifiche del sistema, ovvero ciò che il software deve fare. Questi requisiti definiscono le azioni che il sistema deve essere in grado di svolgere, le operazioni che deve eseguire e i risultati attesi.

RF01:	Requisito autenticazione
Descrizione	L'utente deve potersi autenticare al sistema, per motivi di sicurezza.

Tabella 2.1. Requisito funzionale 01

RF02:	Requisito filtraggio
Descrizione	L'utente deve poter filtrare i dati da visualizzare.

Tabella 2.2. Requisito funzionale 02

RF03:	Requisito salvataggio dati
Descrizione	Il sistema riceve i dati di geocalizzazione.

Tabella 2.3. Requisito funzionale 03

RF04:	Requisito visualizzazione dati
Descrizione	L'utente visualizza i dati geospaziali.

Tabella 2.4. Requisito funzionale 04

2.2.2 Requisiti di sistema

I requisiti di sistema, anche noti come requisiti minimi di sistema o requisiti hardware/-software, sono le specifiche tecniche necessarie affinché un determinato software o sistema operativo funzioni correttamente su un computer o dispositivo. Questi requisiti determinano le capacità hardware e software richieste per l'esecuzione ottimale di un'applicazione o sistema.

RS:	Requisito sistema operativo
Descrizione	Il software sviluppato dovrà essere utilizzato su Windows 11.

Tabella 2.5. Requisito di sistema

2.2.3 Requisiti non funzionali

Specificano attributi di qualità e caratteristiche del sistema che non riguardano direttamente le funzionalità, ma influenzano la sua prestazione, affidabilità, sicurezza, usabilità e altro ancora. Esempi di requisiti non funzionali includono tempi di risposta accettabili, capacità di gestire un certo numero di utenti simultanei, conformità a norme o standard specifici, requisiti di sicurezza e requisiti di usabilità.

RNF01:	Requisiti Prestazionali - Tempi di risposta
Descrizione	Il sistema DEVE consentire all'utente di ottenere risposte in tempo reale. Nel caso in esame, il sistema e' distribuito su un Server, che deve essere accessibile in modalità remota (tramite Web Application). Si richiede, dunque, che per una buona operatività da parte dell'utente è opportuno che il sistema analizzi il carico di rete in modo da scegliere la linea più libera, diminuendo anche la probabilità di perdita di pacchetti dati.

Tabella 2.6. Requisito non funzionale 01

RNF02:	Requisiti Prestazionali - Multiutenza
Descrizione	Il sistema deve garantire che le informazioni trattate dal sistema software devono poter essere gestite da diverse postazioni (terminali).

Tabella 2.7. Requisito non funzionale 02

RNF03:	Requisiti di sicurezza - Integrità dei dati
Descrizione	Il sistema deve garantire l'immutabilità dei dati, assicurando che una volta che i dati sono stati registrati o memorizzati nel sistema, non sia possibile apportarvi modifiche. I dati devono essere trattati come una risorsa in sola lettura e le operazioni di scrittura o modifica dei dati o non concesse affatto.

Tabella 2.8. Requisito non funzionale 03

RNF04:	Vincoli di sistema - Persistenza dei dati
Descrizione	I dati acquisiti dal sistema DOVREBBERO essere memorizzati in maniera persistente.

Tabella 2.9. Requisito non funzionale 04

2.3 Diagramma di contesto

Il diagramma di contesto ha lo scopo di fornire una panoramica chiara e concisa delle interazioni fondamentali tra il sistema e il suo ambiente esterno. Non fornisce dettagli interni del sistema, ma si concentra sulle comunicazioni in ingresso e in uscita tra il sistema e gli attori esterni.

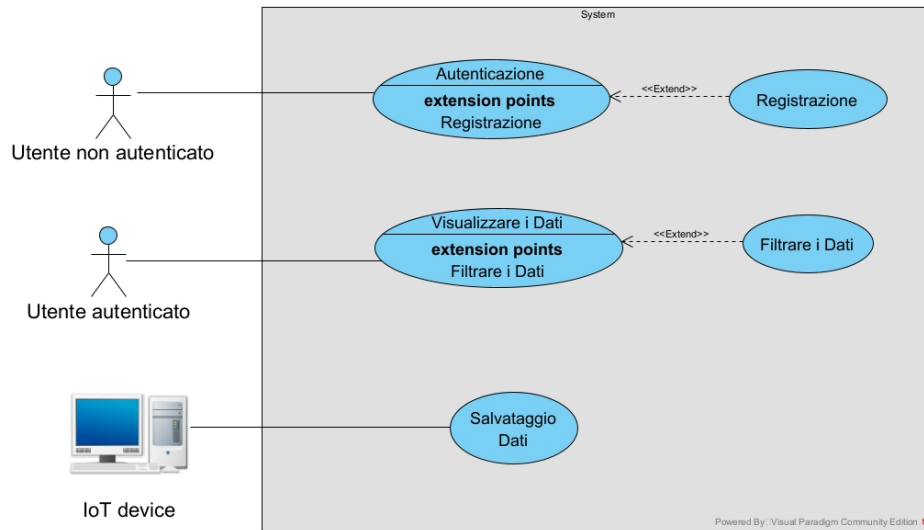


Figura 2.1. Diagramma di contesto

2.4 Casi d'uso

I casi d'uso sono descrizioni delle interazioni tra gli attori (utenti o altri sistemi) e il sistema. Essi rappresentano le funzionalità che il sistema deve fornire per soddisfare le esigenze degli utenti. I casi d'uso sono documentati attraverso descrizioni che includono il nome del caso d'uso, gli attori coinvolti, il flusso principale degli eventi e le possibili estensioni o eccezioni. I casi d'uso sono uno strumento fondamentale per comprendere le interazioni nel sistema, identificare i requisiti funzionali e comunicare in modo chiaro le funzionalità desiderate.

2.4.1 Descrizione dei casi d'uso

Caso d'uso:	Autenticazione
Attore primario	Utente non autenticato
Attore secondario	
Breve descrizione	L'utente deve inserire le proprie credenziali qualora le possieda per visualizzare i dati geospaziali
Pre-condizioni	L'utente non ha già effettuato l'accesso
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'utente inserisce le credenziali di accesso 2. Il sistema controlla se le credenziali hanno una corrispondenza 3. IF non hanno corrispondenza THEN Il sistema reindirizza l'utente alla pagina iniziale ELSE Il sistema chiede all'utente di reinserire le credenziali ripartendo da punto 1.
Post-condizioni	L'utente ha accesso al sistema
Sequenza di eventi alternativi	Se non sono trovate corrispondenza <ol style="list-style-type: none"> 1. Il sistema avvisa l'utente che le sono errate 2. Il sistema richiede di reinserire le credenziali

Tabella 2.10. Requisito di autenticazione

Caso d'uso:	Registrazione
Attore primario	Utente non autenticato
Attore secondario	
Breve descrizione	L'utente inserisce email, password, nome e cognome per registrarsi al servizio.
Pre-condizioni	
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'utente inserisce le credenziali di accesso 2. Il sistema controlla se le credenziali sono già esistenti 3. IF non esistono THEN Il sistema reindirizza l'utente alla pagina iniziale ELSE Il sistema chiede di reinserire nuove credenziali, diverse dalle precedenti, ripartendo da punto 2..
Post-condizioni	L'utente ha accesso al sistema
Sequenza di eventi alternativi	

Tabella 2.11. Requisito di registrazione

Caso d'uso:	Salvataggio dati
Attore primario	IoT device
Attore secondario	
Breve descrizione	Una volta ricevuti i dati geospaziali vengono salvate nel database
Pre-condizioni	Il sistema è pronto a ricevere
Sequenza di eventi principale	1. Il sistema riceve i dati 2. il sistema salva i dati nel database in un apposita tabella
Post-condizioni	Il sistema invia una notifica di operazione andata a buon fine
Sequenza di eventi alternativi	Se non è stato salvato 1. Avvisa il IoT device del fallimento 2. Rimane in attesa di reinvio dei dati

Tabella 2.12. Requisito di salvataggio dati

Caso d'uso:	Visualizzazione dati
Attore primario	Utente autenticato
Attore secondario	
Breve descrizione	L'utente consulta le informazioni contenute nel database.
Pre-condizioni	
Sequenza di eventi principale	1. L'utente richiede i dati 2. Il sistema fornisce i dati richiesti tramite una QUERY
Post-condizioni	I dati sono mostrati a video
Sequenza di eventi alternativi	

Tabella 2.13. Requisito di visualizzazione dati

Caso d'uso:	Filtraggio dati
Attore primario	Utente autenticato
Attore secondario	
Breve descrizione	L'utente filtra le informazioni contenute nel database in base a orario di inizio e fine acquisizione dati.
Pre-condizioni	
Sequenza di eventi principale	1. L'utente inserisce data di inizio e fine acquisizione voluta per il filtro 2. Il sistema filtra i dati richiesti tramite una QUERY 3. Il sistema richiama la funzione Visualizza Dati
Post-condizioni	I dati filtrati sono mostrati a video
Sequenza di eventi alternativi	

Tabella 2.14. Requisito di filtraggio dati

2.5 Stima dei costi

Per la valutazione della stima dei costi si avvale degli Use Case Points(UCP). È una tecnica di stima del software utilizzata per prevedere le dimensioni del software per i progetti di sviluppo software. UCP viene utilizzato quando le metodologie UML (Unified Modeling Language) vengono utilizzate per la progettazione e lo sviluppo del software. Il concetto di UCP si basa sui requisiti per il sistema scritto utilizzando casi d'uso, che fa parte del set UML di tecniche di modellazione. La dimensione del software (UCP) viene calcolata in base agli elementi dei casi d'uso del sistema con il factoring per tenere conto di considerazioni tecniche e ambientali. L'UCP per un progetto può quindi essere utilizzato per calcolare lo sforzo stimato per completare un progetto.

2.5.1 Unadjusted Use Case Weight (UUCW)

I valori di riferimento assegnati a ciascun caso d'uso, detti anche **USE CASE CLASSIFICATION**, sono:

- *Simple* (1 - 3 transiction) Weight 5.
- *Average* (4 - 7 transiction) Weight 10.
- *Complex* (8 or more transiction) Weight 15.

Use case	COMPLESSITÀ
UTENTE NON AUTENTICATO - Autenticazione	Simple
UTENTE NON AUTENTICATO - Registrazione	Simple
IoT DEVICE - Salvataggio dati	Complex
UTENTE AUTENTICATO - Visualizza dati	Simple
UTENTE AUTENTICATO - Filtraggio dati	Average

Tabella 2.15. Elenco casi d'uso Utente, Utente non autenticato, IoT device

Valutazione dell'UUCW:

$$UUCW = (No.of SimpleUC \times 5) + (No.AverageUC \times 10) + (No.ComplexUC \times 15)$$

COMPLESSITÀ	Peso	No. UC:	No.UC * Peso
Simple	5	3	15
Average	10	1	10
Complex	15	1	15
Totale			45

Tabella 2.16. Calcolo UUCW

2.5.2 Unadjusted Actor Weight (UAW)

Valutazione dell'associazione attori del sistema e complessità di realizzazione.

I valori di riferimento:

- Simple: Sistema esterno che deve interagire con il sistema utilizzando un'API ben definita - Weight 1.
- Average: Sistema esterno che deve interagire con il sistema utilizzando protocolli di comunicazione standard (es. TCP/IP, FTP, HTTP, database) - Weight 2.
- Complex: Attore umano che utilizza un'interfaccia applicativa GUI - Weight 3.

Valutazione dell' UAW:

$$UAW = (No.of SimpleActors \times 1) + (No.AverageActors \times 2) + (No.ComplexActor \times 3)$$

ACTORS	COMPLESSITÀ
UTENTE NON AUTENTICATO	Complex
UTENTE AUTENTICATO	Complex
IoT DEVICE	Average

Tabella 2.17. Associazione ACTORS - COMPLESSITÀ

COMPLESSITÀ	Peso	No. Actors:	No. Actors * Peso
Simple	1	0	0
Average	2	1	2
Complex	3	2	6
Totale			8

Tabella 2.18. Calcolo UAW

2.5.3 Technical Complexity Factor (TCF)

Il TCF è uno dei fattori applicati alla dimensione stimata del software al fine di tenere conto delle considerazioni tecniche del sistema. È determinato assegnando un punteggio compreso tra 0 (il fattore è irrilevante) e 5 (il fattore è essenziale) a ciascuno dei 13 fattori tecnici elencati nella tabella seguente. Questo punteggio viene quindi moltiplicato per il valore ponderato definito per ciascun fattore.

I valori di riferimento:

- T1 - Sistema distribuito, 2.0.
- T2 - Tempo di risposta/obiettivi prestazionali, 1.0.
- T3 - Efficienza dell'utente finale, 1.0.
- T4 - Complessità di elaborazione interna, 1.0.
- T5 - Riusabilità del codice, 1.0.
- T6 - Facile da installare, 0.5.
- T7 - Facile da usare, 0.5.
- T8 - Portabilità su altre piattaforme, 2.0.
- T9 - Manutenzione del sistema 1.0.
- T10 - Elaborazione simultanea/parallela, 1.0.
- T11 - Funzioni di sicurezza, 1.0.
- T12 - Accesso per terze parti, 1.0.
- T13 - Formazione dell'utente finale, 1.0.

Per il sistema software in esame si usano i seguenti valori di fattori:

T1 (3), T2 (4), T3 (5), T4 (2), T5 (1), T6 (4), T7 (5), T8 (4), T9 (4), T10 (3), T11 (3), T12 (3), T13 (3).

Il totale di tutti i valori calcolati è il fattore tecnico (TF), calcolato come segue:

$$TF = 3 \times 2 + 4 \times 1 + 5 \times 1 + 2 \times 1 + 1 \times 1 + 4 \times 0.5 + 5 \times 0.5 + 4 \times 2 + 4 \times 1 + 3 \times 1 +$$

$$3 \times 1 + 3 \times 1 + 3 \times 1 = 42.5$$

il TF viene quindi utilizzato per calcolare il TCF con la seguente formula:

$$TCF = 0.6 + (TF/100) = 1.025$$

2.5.4 Environmental Complexity Factor (ECF)

L'ECF è un altro fattore applicato alla dimensione stimata del software al fine di tenere conto delle considerazioni ambientali del sistema. È determinato assegnando un punteggio compreso tra 0 (nessuna esperienza) e 5 (esperto) a ciascuno degli 8 fattori ambientali elencati nella tabella seguente. Questo punteggio viene quindi moltiplicato per il valore ponderato definito per ciascun fattore.

Valori di riferimento:

- E1 - Familiarità con il processo di sviluppo utilizzato, 1.5.
- E2 - Esperienza applicativa, 0.5.
- E3 - Esperienza di team orientata agli oggetti, 1.0.
- E4 - Capacità di lead analyst, 0.5.
- E5 - Motivazione del team, 1.0.
- E6 - Stabilità dei requisiti, 2.0.
- E7 - Personale part-time, 0.0.
- E8 - Linguaggio di programmazione difficile, 0.0.

Per il sistema software in esame si usano i fattori:

$$E1 (3) + E2 (4) + E3 (2) + E4 (3) + E5 (5) + E6 (4) + E7 (1) + E8(3).$$

Il totale di tutti i valori calcolati 'e il fattore ambiente (EF), calcolato come segue:

$$EF = 3 \times 1.5 + 4 \times 0.5 + 2 \times 1 + 3 \times 0.5 + 5 \times 1 + 4 \times 2 + 1 \times (0) + 3 \times (0) = 23$$

L'EF viene quindi utilizzato per calcolare l'ECF con la seguente formula:

$$ECF = 1.4 + (-0.03 \times 23) = 0.71$$

2.5.5 Total Use Case Points (UCP)

Infine, una volta determinate le dimensioni del progetto non aggiustate (UUCW e UAW), e calcolati il fattore tecnico (TCF) e il fattore ambientale (ECF) si pu'ò calcolare l'UCP. L'UCP viene calcolato in base alla seguente formula:

$$UCP = (UUCW + UAW) \times TCF \times ECF = (45 + 10) \times 1.025 \times 0.71 \approx 40$$

Da quest'ultimo parametro è possibile valutare il numero di ore di lavoro totali considerando che un singolo UC venga sviluppato in 8 ore mediamente:

$$Th = UCP \times 8 = 40 \times 8 = 320h$$

In definitiva, se si suppone che un membro del progetto lavori per circa 40 ore a settimana

(Wh), ed il team sia composto da 4 membri, essi lavoreranno per un totale di 160 ore a settimana (TWh). Th è il numero di ore di lavoro totali necessarie a terminare il progetto e TWh il numero di ore di lavoro a settimana del gruppo di lavoro (team), allora il numero di settimane necessarie per terminare il lavoro è dato da:

$$Th/TWh = 320/160 = 2$$

Dunque si stima che occorreranno circa 2/3 settimane per sviluppare l'intero progetto.

2.6 System Domain Model

Il System Domain Model (SDM) è un modello concettuale che astrae e rappresenta gli elementi e/o entità del sistema software che si vuole realizzare (dominio), descrivendo le relazioni che intercorrono tra di essi, i possibili ruoli e attributi. Tale modello dà una descrizione della struttura (quindi una descrizione statica) del sistema calato nel dominio operativo. Utilizziamo il dataframe per salvare le informazioni estratte dal file UBX

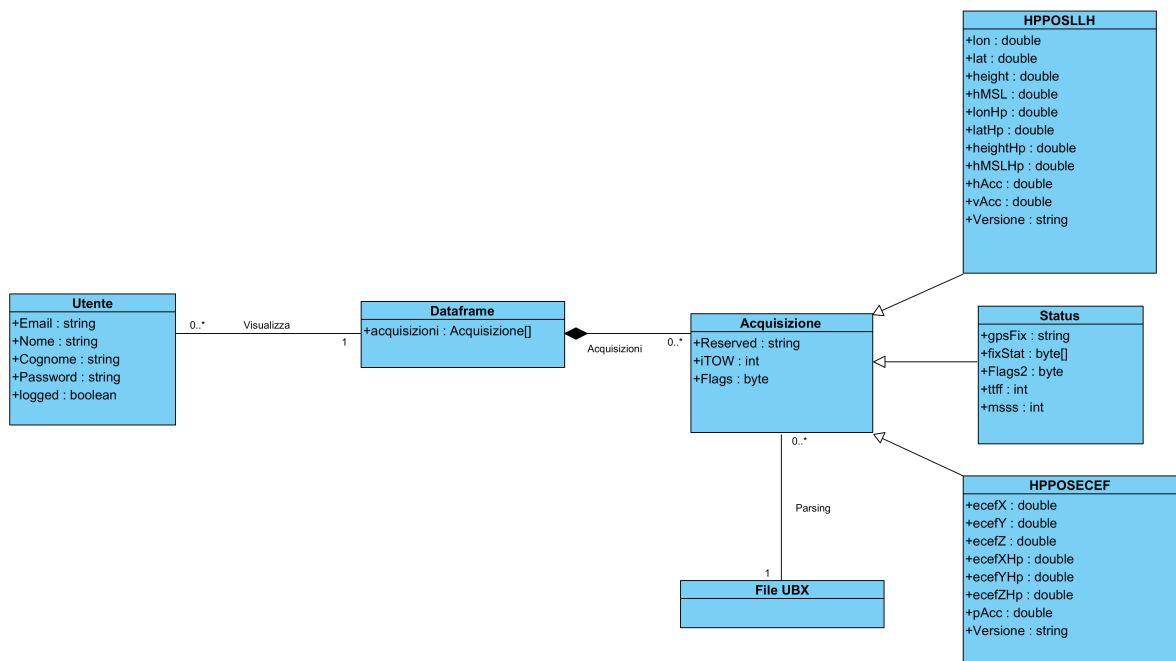


Figura 2.2. Diagramma di SDM

2.7 Diagrammi di sequenza

Questi diagrammi rappresentano l'interazione tra gli oggetti all'interno di un sistema in un determinato scenario o flusso di esecuzione. Attraverso la sequenza di messaggi e attivazioni, i diagrammi sequenziali mostrano l'ordine in cui gli oggetti interagiscono e si scambiano informazioni nel sistema. Consentono di visualizzare l'interazione dinamica tra gli oggetti durante l'esecuzione di uno scenario specifico.

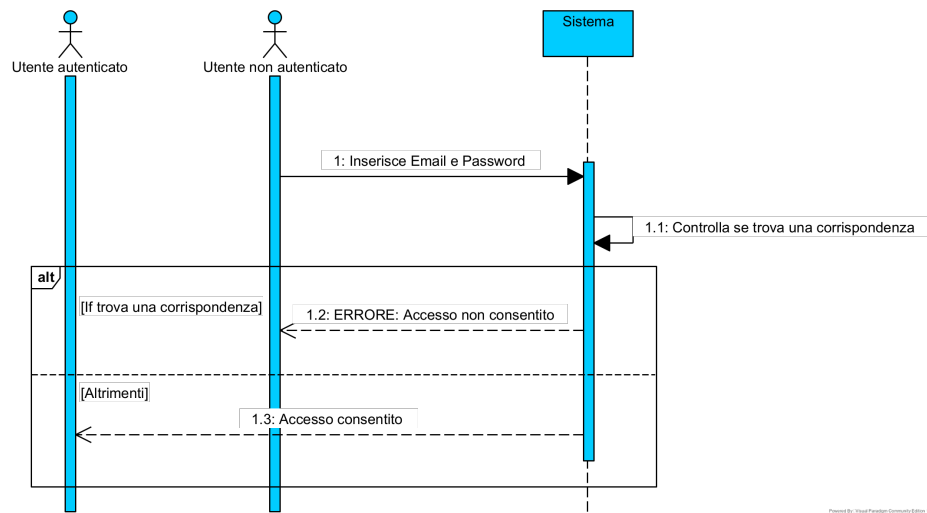


Figura 2.3. Sequenza di login

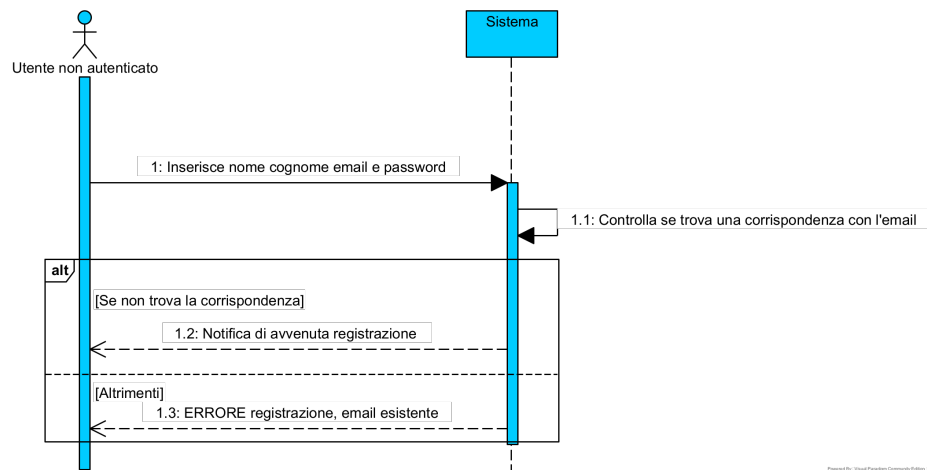


Figura 2.4. Sequenza di registrazione

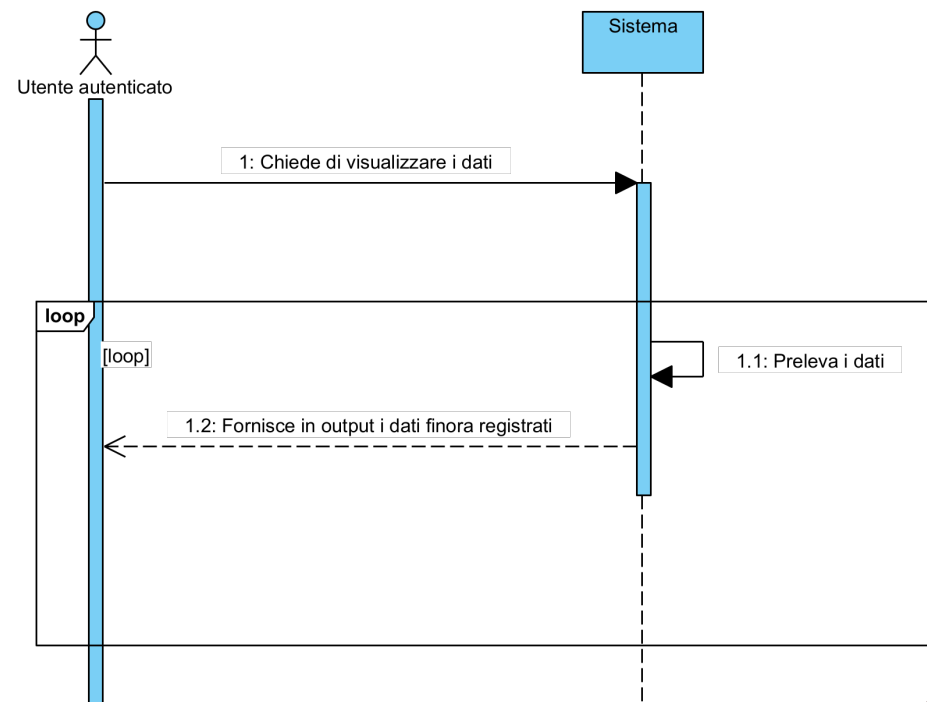


Figura 2.5. Sequenza di acquisizione dati

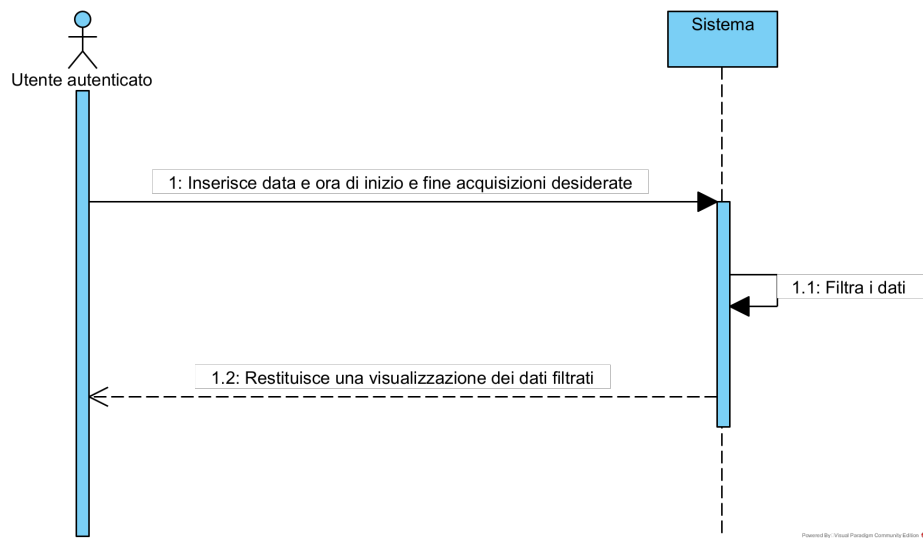


Figura 2.6. Sequenza di filtraggio dati

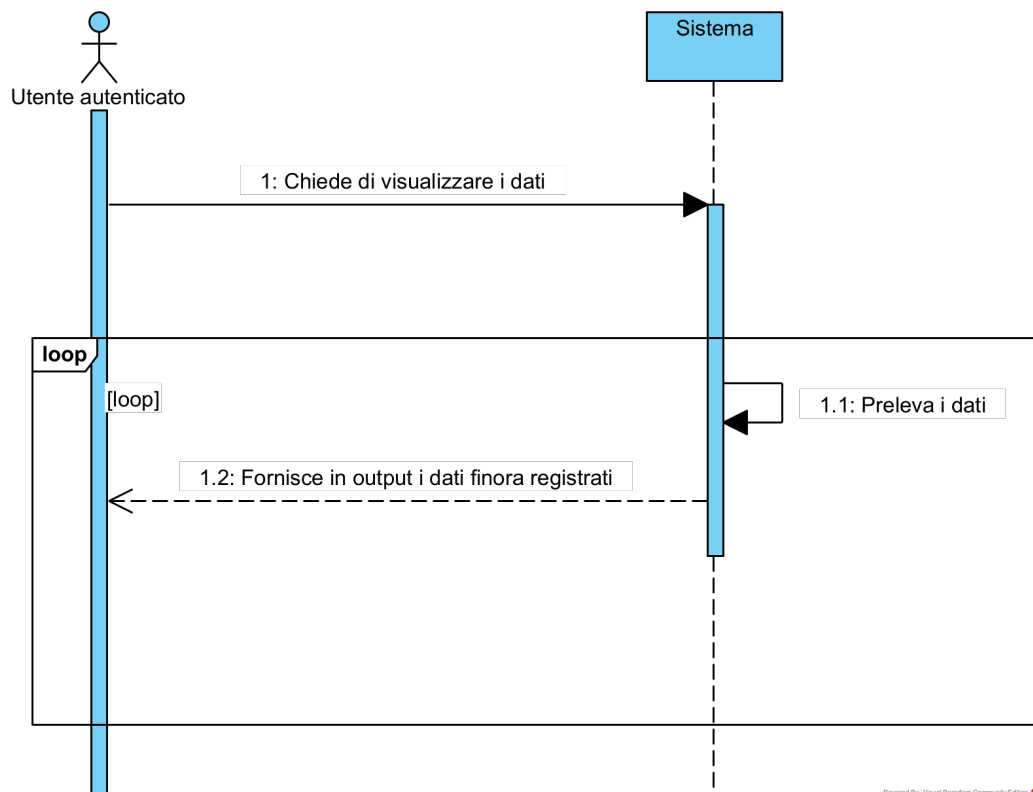


Figura 2.7. Sequenza di visualizza dati

Progettazione

Il presente capitolo si focalizza sull'analisi delle specifiche progettuali del sistema, definendo i componenti, i connettori e i pattern architetturali chiave utilizzati.

3.1 Scelte progettuali

Per definire le scelte progettuali del prodotto software in questione, si delineano gli stili e i pattern che definiranno il progetto. Le scelte saranno:

- **Pattern Architetturale:** Client-server, in quanto c'è la necessità di separare la visualizzazione dei dati, dalla loro allocazione fisica. E' ottimale per modellare le specifiche richieste in quanto, a differenza ad esempio ad una soluzione architetturale MVC (Model-View-Controller), necessita di una logica di controllo molto basilare. Scegliere di utilizzare una soluzione MVC in questo caso andrebbe infatti soltanto ad aumentare inutilmente la complessità del sistema.
- **Stile Architetturale:** Service-oriented, è stato scelto questo stile per
 - Isolare le responsabilità del sistema a sè stesso;
 - Garantire l'indipendenza e l'autonomia del sistema da moduli esterni;
 - Essere riutilizzabile
 - Essere facilmente modificabile, perchè tutte le modifiche apportate al sistema non intralciano i moduli esterni con cui comunica.
- **Design Pattern:** Proxy, per migliorare la scalabilità e la modularità del sistema. Il Proxy può implementare logiche aggiuntive per controllare l'accesso all'oggetto reale

3.2 Pattern Architetturale - Client-Server

Come definito in precedenza, la scelta del Pattern Architetturale è ricaduta sul client server. Consente la realizzazione di un'architettura che permette la separazione della visua-

lizzazione dati (**client**) e la gestione di servizi (**server**). Lo schema di Alto livello del Pattern Architeturale è rappresentato in figura 3.1.

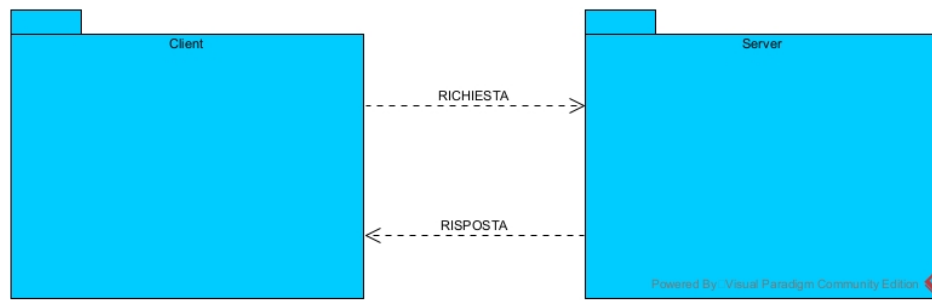


Figura 3.1. Diagramma Client-Server

3.2.1 Client

Il client è una componente di un'applicazione client-server ed è responsabile dell'interazione con l'utente e dell'invio delle richieste al server per ottenere i dati o eseguire operazioni specifiche.

3.2.2 Server

Il server è una componente di un'applicazione client-server ed è responsabile di gestire le richieste inviate dal client, elaborarle e fornire una risposta appropriata.

3.3 Stile Architeturale - Service-oriented (SOA)

Il sistema viene progettato ed implementato come un servizio che parli con il computer utilizzando un protocollo REST.

- **Decomposizione dei servizi:** L'applicazione viene scomposta in servizi autonomi che possono essere sviluppati, testati e aggiornati indipendentemente l'uno dall'altro. Questa modularità favorisce il riutilizzo del codice e semplifica la manutenzione.
- **Interoperabilità:** I servizi possono essere sviluppati utilizzando tecnologie e linguaggi di programmazione diversi, purché siano in grado di comunicare tramite i protocolli definiti. Ciò consente di integrare e far interagire sistemi eterogenei.
- **Standardizzazione:** SOA fa ampio uso di standard aperti e protocolli comuni per la comunicazione tra i servizi. Ciò facilita l'interoperabilità e riduce la dipendenza da tecnologie proprietarie.
- **Scalabilità:** I servizi possono essere scalati indipendentemente l'uno dall'altro in base alle esigenze di carico. Ciò consente di gestire carichi di lavoro elevati senza dover scalare l'intera applicazione.

- Riusabilità: I servizi possono essere progettati in modo da poter essere riutilizzati in diverse applicazioni o contesti. Ciò promuove l'efficienza nello sviluppo del software e riduce la duplicazione del codice.
- Gestione dei processi: SOA consente di definire e gestire i flussi di lavoro e i processi aziendali come servizi separati. Ciò favorisce l'automazione dei processi e l'integrazione con i sistemi esistenti.

3.4 REST

Vale la pena dunque definire REST (REpresentational State Transfer). Esso pone dei requisiti architetturali alla quale il sistema deve soddisfare, utilizzando degli standard per la rappresentazione dei dati e l'HTTP per il trasferimento di questi dati o altri. Quando il Client chiama un'API RESTful, il Server trasferirà al Client una rappresentazione dello stato della risorsa richiesta.

Quando il Client richiede al Server una risorsa, dovrà fornirgli un'identificatore per la risorsa che gli interessa (URL) e l'operazione che si desidera che il Server esegua su tale risorsa sotto forma di metodi HTTP:

- POST: crea una nuova risorsa;
- GET: restituisce una o più risorse esistenti;
- PUT: aggiorna una risorsa o, se non esiste, la crea;
- DELETE: elimina una risorsa.

Vantaggi della soluzione REST: Separazione tra Client e Server che consente di sviluppare i diversi componenti in maniera indipendente tra loro, alta scalabilità dovuta al fatto che Server e Client possono risiedere su macchine diverse, indipendenza delle API REST dal linguaggio o dalle tecnologie utilizzate per implementare i componenti.

Riguardo quest'ultimo punto bisogna, per ogni linguaggio, mantenere solamente la coerenza sul formato delle richieste e delle risposte che ci si aspetta dal Client o dal Server in base a come esse siano state definite. Nel paragrafo successivo si elencano i principi base da seguire per costruire un'architettura REST affinché le API siano definite RESTful.

3.4.1 Client - Server in REST

Basato sul paradigma SoC (Separation of Concerns), questo principio stabilisce la separazione dei compiti tra Client e Server. Infatti, in un'architettura distribuita, bisogna definire le figure di Client e Server che avranno il compito di svolgere una determinata funzionalità. Si ha quindi che il Client invocherà, tramite un messaggio di richiesta, un servizio esposto dal Server, che a sua volta si occuperà di elaborare la richiesta e di fornire una risposta al Client.

Come è formata una richiesta:

- Endpoint: URL in cui il Server REST è in ascolto.
- Metodo: utile per permettere di richiedere dati o modificarli in modo tale che il Server sappia quale operazione è stata richiesta dal Client. Come già detto in precedenza, i metodi sono GET, POST, PUT, DELETE e altri.
- Intestazione: i dettagli aggiuntivi forniti per la comunicazione tra Client e Server (si ricorda che REST è senza stato). Alcune delle intestazioni comuni sono:
 - Request:
 - * host: IP del Client
 - * accept-language: lingua comprensibile dal cliente
 - * user-agent: informazioni relative al Client (sistema operativo, etc...)
 - Response:
 - * status: lo stato della richiesta o codice HTTP
 - * content-type: tipo di risorsa inviata dal Server.
 - * set-cookie: imposta i cookie per il Server
- Dati: contiene le informazioni che si desiderano inviare al Server (chiamato anche corpo o messaggio).

3.4.2 Stateless

La comunicazione tra il Client e il Server deve essere "senza stato", cioè la richiesta del Client verso il Server deve contenere tutte le informazioni utili in modo tale che essa possa essere soddisfatta sul servizio chiamato. Viene quindi eliminata la correlazione tra le varie richieste, facendo così apparire ogni richiesta come se fosse la prima verso quel Server. Inoltre, questo tipo di comunicazione dà la possibilità di poter scalare il sistema molto più facilmente, poiché non vi è più il problema di sincronizzare le informazioni sulla sessione degli utenti tra i vari nodi del Server; questa informazione viene conservata lato Client oppure demandata a un database.

3.4.3 Cacheable

Le risposte fornite dal Server possono essere etichettate come cachabili o non cachabili. Nel caso di risposte cachabili, le informazioni contenute al suo interno possono essere riutilizzate nelle richieste successive evitando l'interazione con il Server poichè esse sono informazioni ancora valide ed attuali. Questo porta ad un conseguente miglioramento della scalabilità e delle performance del Server. Contrariamente, le risposte possono essere definite non cachabili se contengono informazioni su stati che in futuro potrebbero non essere più validi.

3.4.4 Interfaccia uniforme

La comunicazione tra Client e Server deve avvenire tramite un'interfaccia uniforme che preveda la definizione delle risorse e del formato nella quale esse devono essere rappresentate. Una risorsa 'è l'elemento fondamentale del servizio REST: essa rappresenta un oggetto identificante un qualcosa di inerente al dominio applicativo al quale si accede tramite un identificatore globale (URI) e di cui Client e Server si scambiano rappresentazioni tramite un'interfaccia standard come HTTP. La rappresentazione più usata per le risorse 'è il formato JSON, ma vi sono altri formati utilizzabili come HTML come utilizzato per questo progetto.

3.4.5 Sistema stratificato

Il sistema stratificato permette ad un'architettura di essere composta da più livelli intermedi posti tra Client e Server, indipendenti tra loro ed ognuno dei quali svolge un determinato compito. Ad esempio, strati intermedi possono essere composti da sistemi che svolgono azioni di caching, di sicurezza o di load-balancing. Il vantaggio di avere un sistema stratificato con livelli indipendenti sta nella possibilità di poter aggiungere, cancellare o spostare i livelli intermedi in base al comportamento che si vuole ottenere nell'architettura, senza influenzare gli altri livelli già presenti.

3.5 Diagramma dei componenti

Si rappresenta inizialmente la struttura interna del sistema software in termini dei suoi componenti principali e delle relazioni fra di essi. I componenti forniscono funzionalità specifiche per il prodotto software che si sta realizzando mentre i connettori forniscono meccanismi di interazione indipendenti dall'applicazione. Con questo diagramma (Figura 3.2) si cerca di separare l'elaborazione dall'interazione esplicitandone i collegamenti che vi sono tra i componenti.

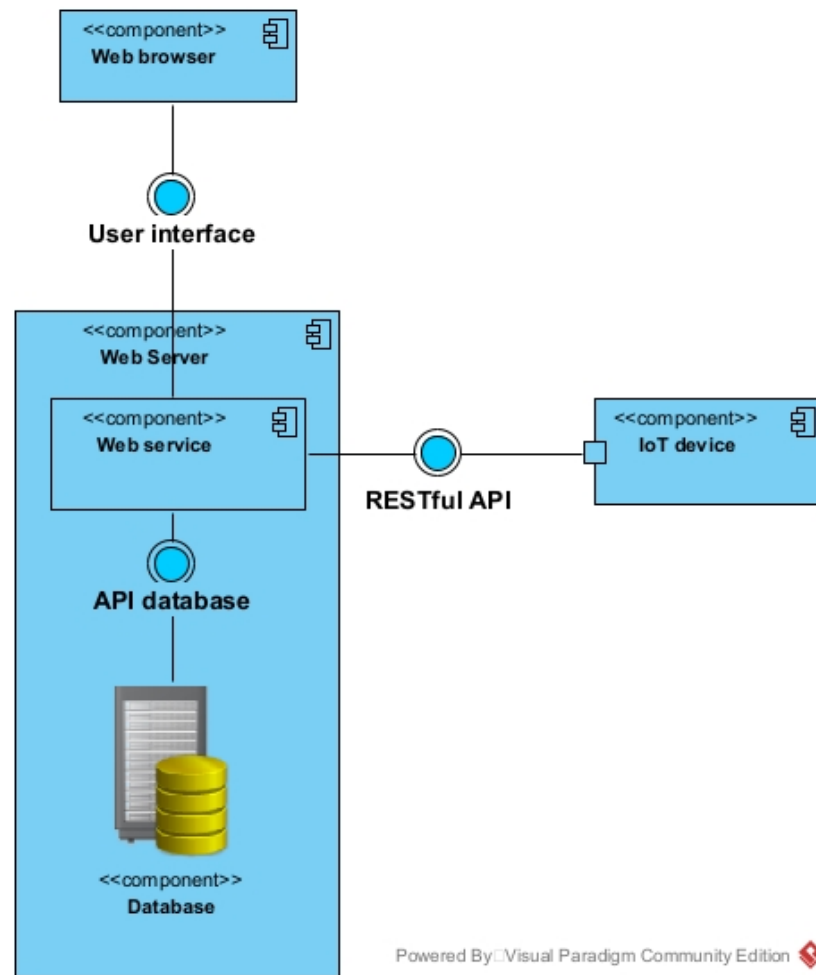


Figura 3.2. Diagramma dei componenti e connettori

3.6 Diagramma delle classi

Modelliamo adesso la componente legata al Web Server, attraverso un diagramma delle classi raffinato.

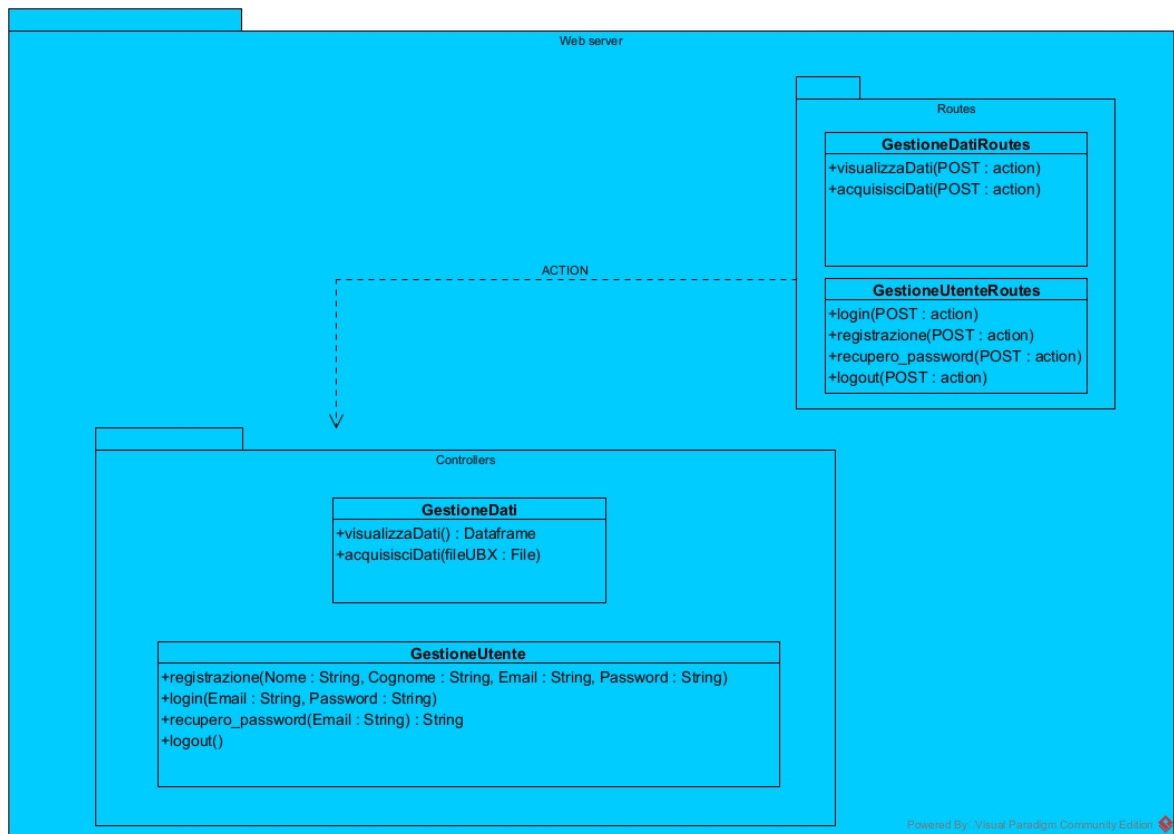


Figura 3.3. Diagramma delle classi

Abbiamo racchiuso le funzionalità del Web Server in due macrogruppi:

- Controller -> Responsabile dell'interfacciamento con i componenti esterni e della modifica del flusso di dati.
- Routes -> Responsabile della gestione dei flussi di controllo.

3.7 Diagrammi di sequenza di progettazione

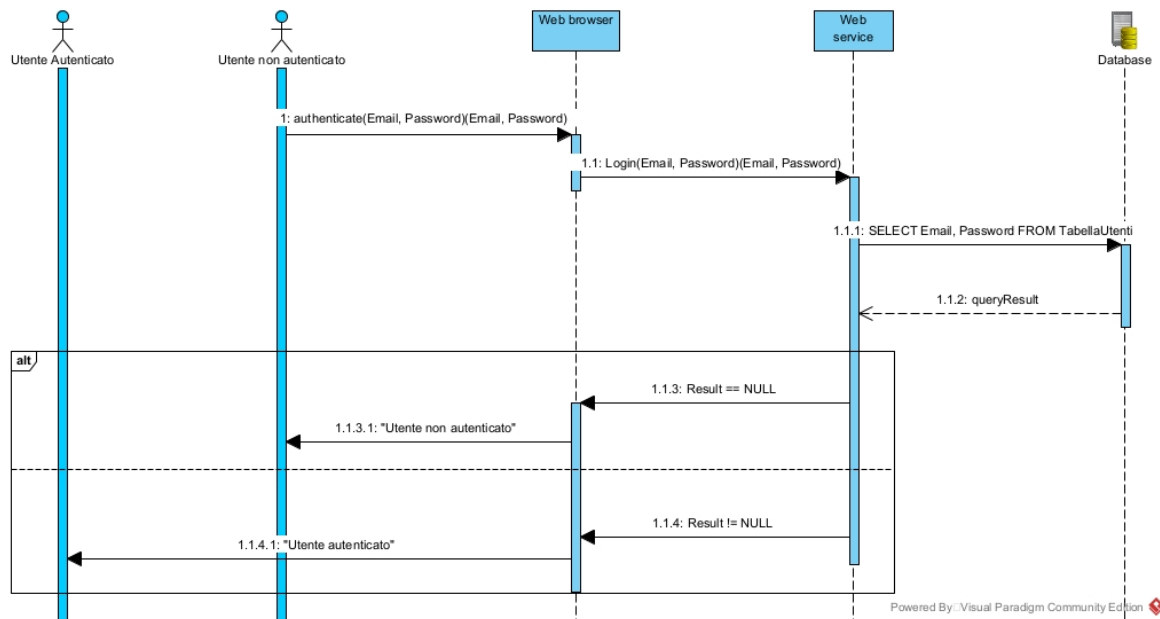


Figura 3.4. Diagramma login

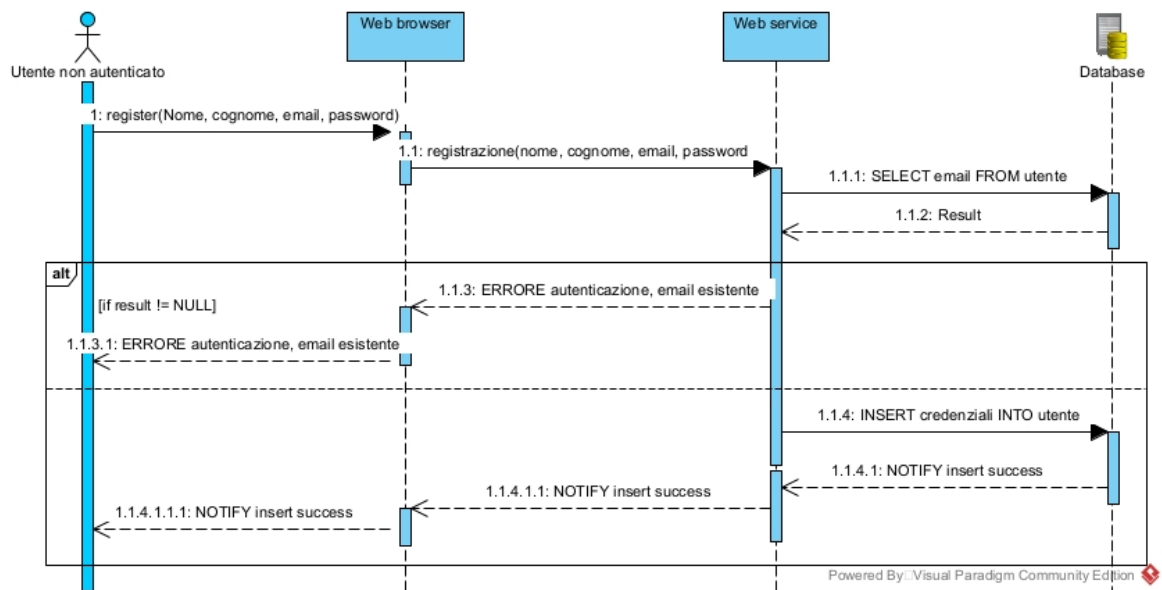


Figura 3.5. Diagramma registrazione

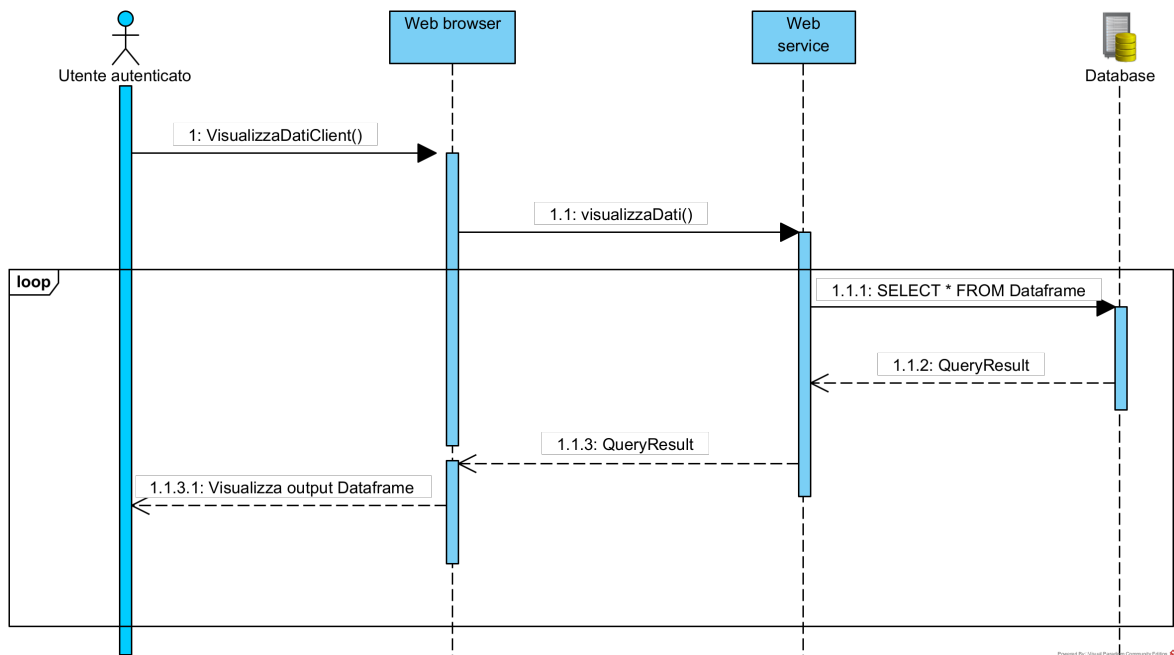


Figura 3.6. Diagramma visualizza dati

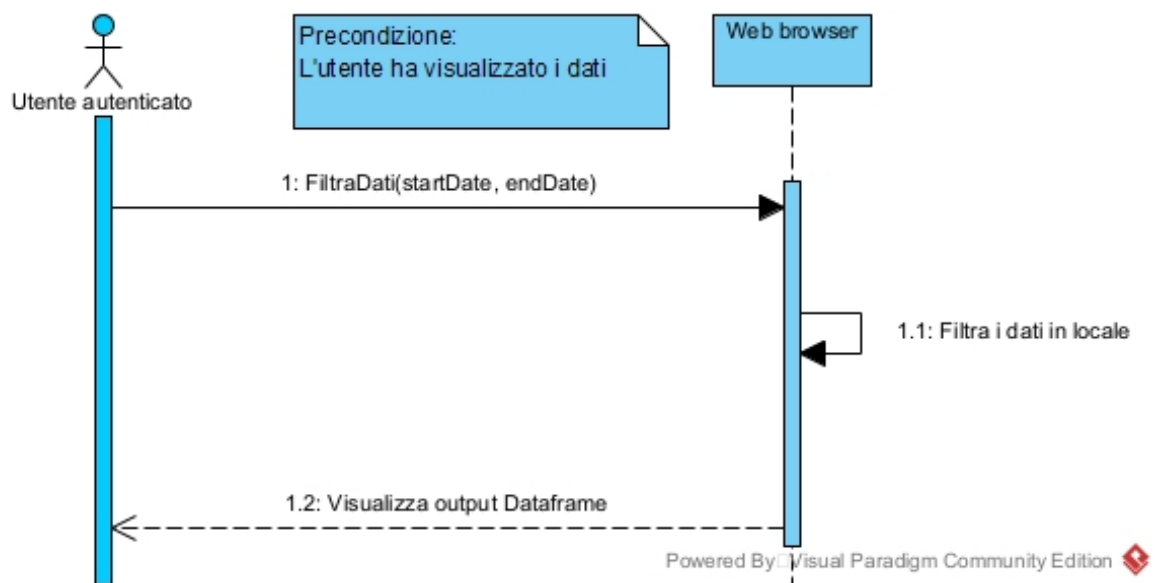
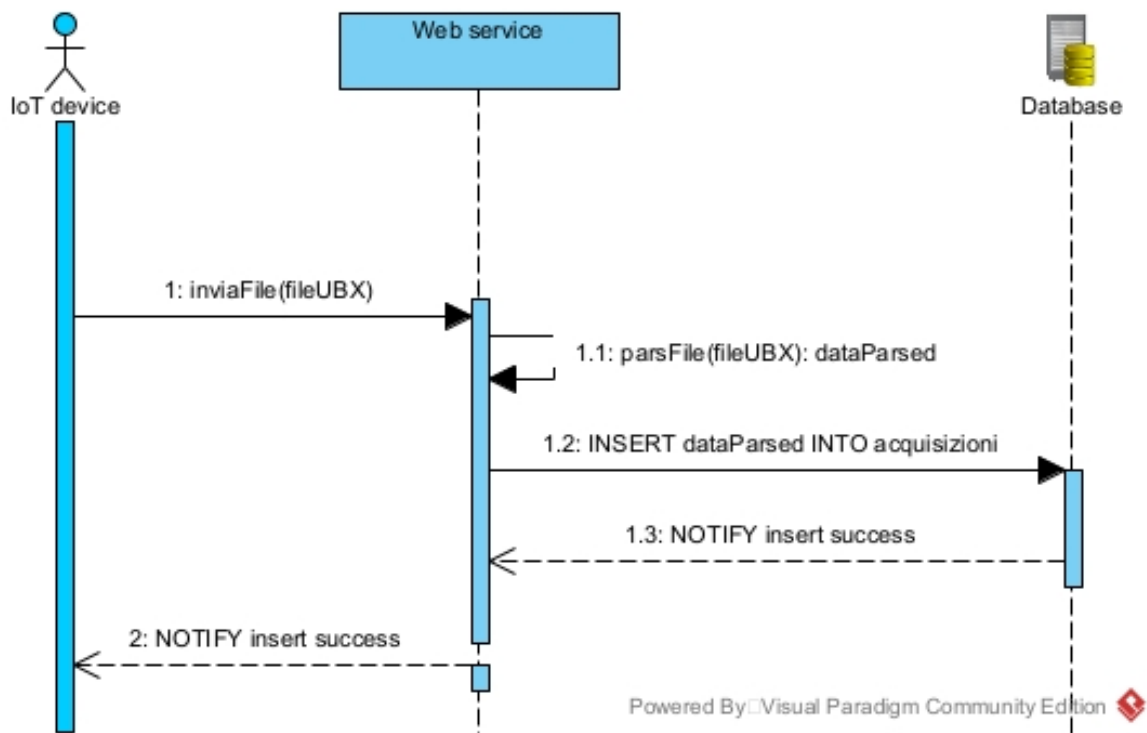
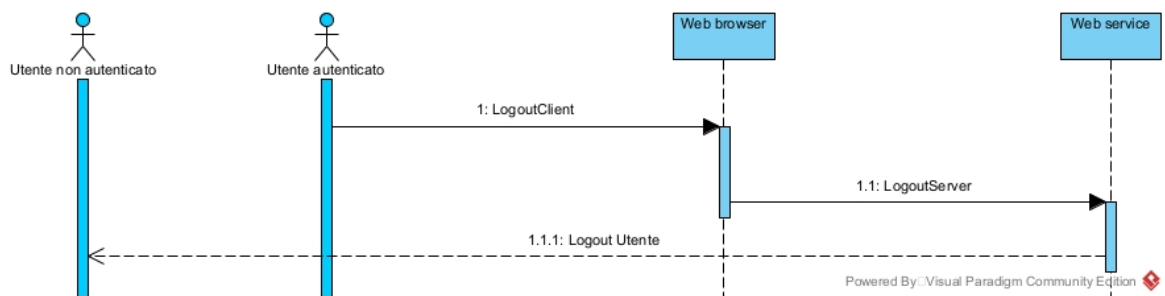


Figura 3.7. Diagramma filtra dati

**Figura 3.8.** Diagramma acquisizione dati**Figura 3.9.** Diagramma logout

3.8 Diagramma di deployment

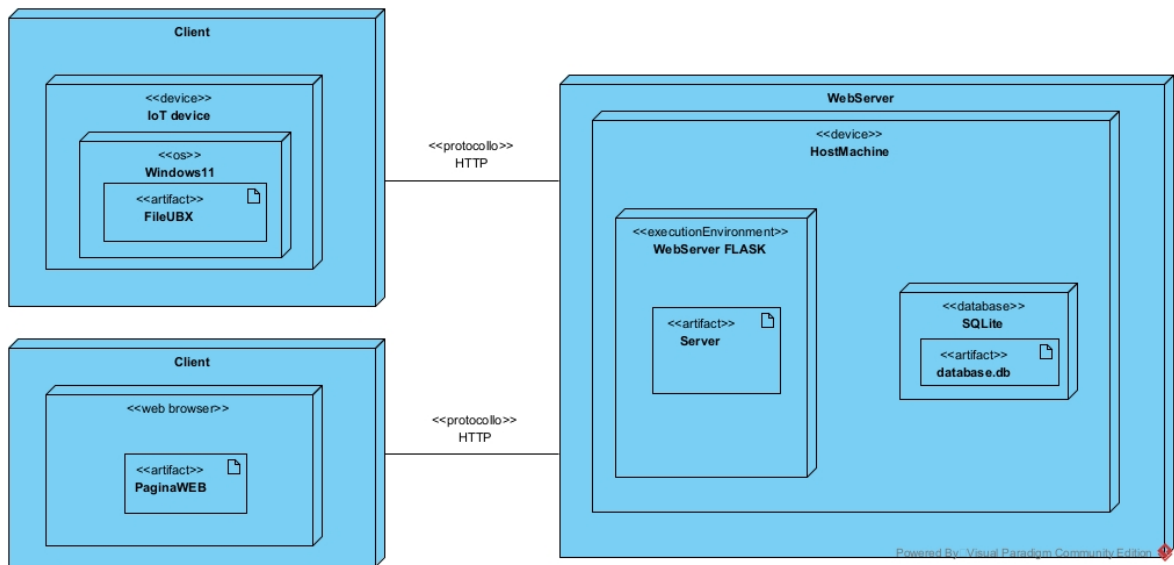


Figura 3.10. Deployment diagram

Implementazione

4.1 Scelte di sviluppo

Le scelte di sviluppo del prodotto software sono state influenzate dal garantire la semplicità di supporto per il web server. Essa comprende la parte relativa all' interfaccia visibile all' utente con la quale gli utenti interagiscono, definendo le funzionalità degli elementi visibile e le loro interazioni con il back-end; quest' ultimo contiene tutte le funzioni che svolge il Server, come ad esempio il salvataggio e la visualizzazione dei dati. La si divide in livelli logico funzionali (in tre livelli: Figura 4.1):

- **Livello di Presentazione:** associabile al front-end, rappresenta l'interfaccia dell'applicazione con cui l'utente interagisce per immettere i dati e leggere i risultati attraverso il Web Browser (User Interface visualizzata tramite pagine HTML,CSS,Javascript);
- **Livello server:** associabile al back-end, rappresenta la logica di elaborazione poichè elabora i dati e in base alle richieste dell'utente genera dei risultati (WebServer con i relativi moduli della web application). Inoltre, interagisce con il database;

Il client e server sono moduli indipendenti, per cui una modifica a uno dei livelli non si ripercuote direttamente su un altro livello. Parlando più specificamente del front-end, esso è costruito usando linguaggi come HTML, CSS e JavaScript, i quali sono supportati da gran parte dei browser.

4.2 Framework di sviluppo

Per l'implementazione del Web server si è utilizzando il framework web FLASK. Esso è utilizzato anche per il front-end integrandolo con dei moduli HTML. Inoltre, si è utilizzato il framework Twilio Sendgrid per l'invio di email.

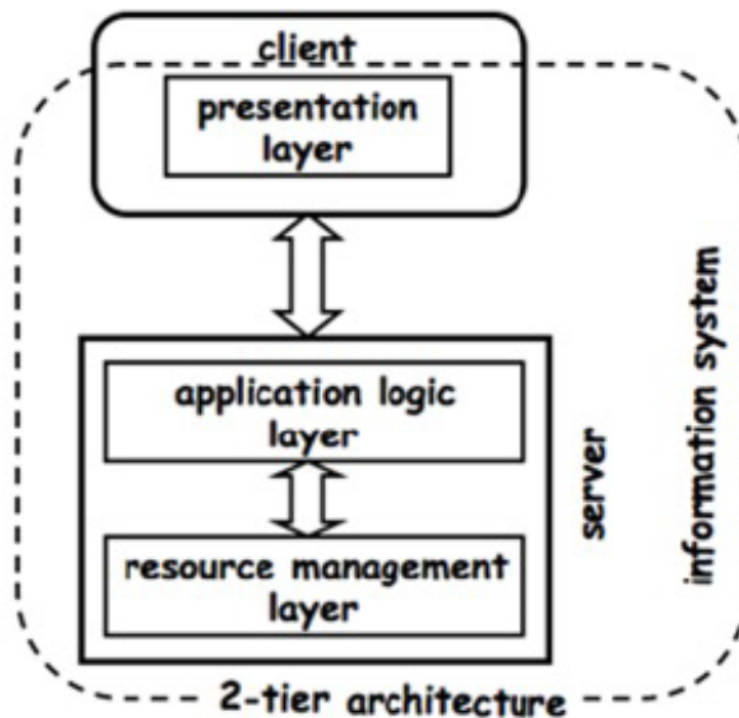


Figura 4.1. Architettura 2-tiers

4.2.1 Back-end: FLASK

Flask offre agli sviluppatori varietà di scelta durante lo sviluppo di applicazioni WEB, fornisce strumenti, librerie e meccanismi che consentono di creare applicazioni senza dipendenze esterne. Flask è basato sul toolkit Werkzeug WSGI e sul motore di template Jinja2. In particolare:

- **WSGI:** Web Server Gateway Interface è un'interfaccia utilizzata come standard per lo sviluppo di applicazioni Web Python. È un protocollo di trasmissione che stabilisce e descrive comunicazioni ed interazioni tra Server ed applicazioni web scritte nel linguaggio Python.
- **Werkzeug:** Werkzeug è un toolkit WSGI che implementa richieste, oggetti di risposta e funzioni di utilità. Ciò consente di costruire un frame Web su di esso.
- **Jinja2:** è un motore di modelli per Python. Un sistema di modelli Web che combina un modello con un'origine dati specifica per eseguire il rendering di una pagina Web dinamica.

Per entrare più nel dettaglio ci sono due lati coinvolti nel rispondere alla richiesta HTTP di un Client: il Server Web e l'applicazione Web.

- Il Server gestisce la complessità delle connessioni di rete, ovvero, riceve la richiesta dal Client e invia la risposta.

- L'applicazione, prende i dati della richiesta, agisce su di essi e crea la risposta per il Server da inviare. Cioè l'applicazione, contiene quella funzione che prende i dati input, esegue la procedura e rimanda in output la risposta da inviare.

L'interazione tra il Server Web e l'applicazione Web è WSGI (Web Server Gateway Interface), ma per garantire che si stia realizzando un'applicazione Web Python, quest'ultima deve assicurarsi che le funzioni create accettino determinati parametri indicati nell'intestazione HTTP e i dati del modulo di input. Se invece si vuole un Server Web che serva applicazioni Python, allora quest'ultimo deve essere in grado di richiamare quella funzione nell'applicazione web ogni qual volta che arriva una richiesta HTTP. WSGI specifica esattamente quali devono essere i parametri in input per quella funzione e restituisce al Server l'output inviato da quella funzione. Werkzeug fornisce una serie di utility per lo sviluppo di applicazioni conformi a WSGI. Queste utility sono l'analisi dell'intestazioni, l'invio e la ricezione di cookie, l'accesso ai dati del modulo, la generazione di reindirizzamenti, la generazione di pagine di errore quando c'è un'eccezione, persino la fornitura di un Server per un debugger interattivo che viene eseguito nel browser. Per completare dunque l'applicazione, serve comunque un template che possa essere presentabile e interattivo all'utente. Jinja2 può essere integrato nei formati di markup (HTML nel caso in esame) in modo da restituire all'utente tramite una richiesta HTTP una pagina web dinamica. È detto motore di modello in quanto esso permette di poter aggiungere tra i tag HTML variabili che vengono sostituite dai valori passati dopo aver eseguito il rendering della pagina HTML.

Il particolare vantaggio di Jinja2 in HTML è quello di poter riuscire ad utilizzare statements ciclici come il for, oppure condizionali come l'if e tante altre skill che essa mette a disposizione nella sua documentazione. Questo permette dunque di avere dinamicità nelle pagine HTML.

In definitiva i vantaggi del framework rispettano le esigenze agili del prodotto software in esame. Esso infatti è scalabile quindi si ha la possibilità di incrementare rapidamente in qualsiasi direzione (verticale o orizzontale) consentendo di funzionare senza problemi anche mentre si espande e aumenta (approccio adattivo), è leggero quindi non si basa su un gran numero di estensioni per funzionare, c'è più controllo sotto questo aspetto, è flessibile e molto documentato dunque un ottimo supporto durante lo sviluppo dell'applicativo.

4.2.2 Gestione mail: SENDGRID

SendGrid è un potente servizio di invio di email basato su cloud che fornisce un'API semplice ed efficiente per inviare email transazionali e di marketing. È una soluzione altamente affidabile e scalabile, utilizzata da migliaia di aziende in tutto il mondo per automatizzare l'invio di email personalizzate e coinvolgenti.

SendGrid semplifica l'invio di email attraverso un'interfaccia semplice ed intuitiva, permettendo agli sviluppatori di integrare facilmente la funzionalità di invio email all'interno delle loro applicazioni Python.

Con la libreria ufficiale "sendgrid" per Python, è possibile inviare email personalizzate con facilità, consentendo una comunicazione efficace con gli utenti.

SendGrid offre i seguenti vantaggi:

- **Affidabilità:** SendGrid garantisce una consegna rapida ed efficiente delle email grazie alla sua infrastruttura cloud robusta.
- **Scalabilità:** Il servizio è in grado di gestire volumi elevati di email senza problemi, adattandosi alle esigenze del tuo progetto.
- **Personalizzazione:** Puoi inviare email personalizzate utilizzando template predefiniti o modelli personalizzati, offrendo contenuti rilevanti e coinvolgenti ai destinatari.
- **Monitoraggio avanzato:** SendGrid fornisce funzionalità di tracciamento delle email inviate, consentendoti di monitorare la consegna, l'apertura e i clic per valutare l'efficacia delle tue campagne.
- **Sicurezza:** SendGrid implementa misure di sicurezza avanzate per proteggere le email e garantire che raggiungano i destinatari senza problemi.

SendGrid è stata utilizzata nel progetto per implementare il metodo "Recupera password". Questo metodo consente agli utenti di recuperare la propria password tramite l'invio di una email all'indirizzo di registrazione.

Grazie a SendGrid, l'utente riceverà un'email contenente le informazioni per il recupero della password.

4.3 Pattern Proxy

Nel progetto è stato utilizzato il pattern Proxy, esso è un design pattern strutturale che offre un livello di astrazione e controllo aggiuntivo tra il client e l'oggetto reale. Questo pattern viene utilizzato quando si desidera fornire un'interfaccia sostitutiva o proxy per l'accesso a un oggetto complesso o risorsa costosa da creare o gestire direttamente.

Il Proxy agisce come un intermediario tra il client e l'oggetto reale, fornendo un'interfaccia simile o identica a quella dell'oggetto reale. Ciò consente al client di interagire con il Proxy come se fosse l'oggetto reale stesso, senza conoscere i dettagli di implementazione o l'ubicazione effettiva dell'oggetto.

Una delle principali utilità del Proxy è il controllo dell'accesso. Il Proxy può gestire le autorizzazioni e il controllo dell'accesso all'oggetto reale, consentendo di applicare restrizioni o regole di sicurezza. Ad esempio, il Proxy può verificare se il client ha i privilegi necessari prima di consentire l'accesso all'oggetto reale.

Un'altra funzionalità del Proxy è l'ottimizzazione delle prestazioni. Il Proxy può memorizzare in cache i risultati delle operazioni costose effettuate sull'oggetto reale. Quando il client richiede nuovamente la stessa operazione, il Proxy può restituire il risultato memorizzato dalla cache senza dover eseguire nuovamente l'operazione sull'oggetto reale. Ciò contribuisce a migliorare le prestazioni e ridurre il carico sul sistema.

In sintesi, il pattern Proxy offre un livello di astrazione e controllo aggiuntivo tra il client e l'oggetto reale, consentendo di gestire l'accesso, ottimizzare le prestazioni e semplificare l'utilizzo delle risorse del sistema.

4.4 Entity Relationship Diagram

Il database è stato progettato a partire dal system domain model, identificando e modellando le entità coinvolte nel sistema. Durante il processo di progettazione, sono stati applicati raffinamenti per ottimizzare la struttura del database.

In particolare, attraverso questi raffinamenti, le relazioni di generalizzazione sono state eliminate. Questo è stato fatto per evitare ridondanze e migliorare l'efficienza dell'interrogazione del database tramite query. Le entità sono state modellate in modo più specifico e dettagliato, consentendo una migliore organizzazione dei dati e una maggiore efficienza nelle operazioni di ricerca e recupero delle informazioni.

Grazie a questi raffinamenti, il database offre una base solida e efficiente per l'interrogazione tramite query, garantendo prestazioni ottimali e riducendo la complessità delle operazioni di recupero dei dati.

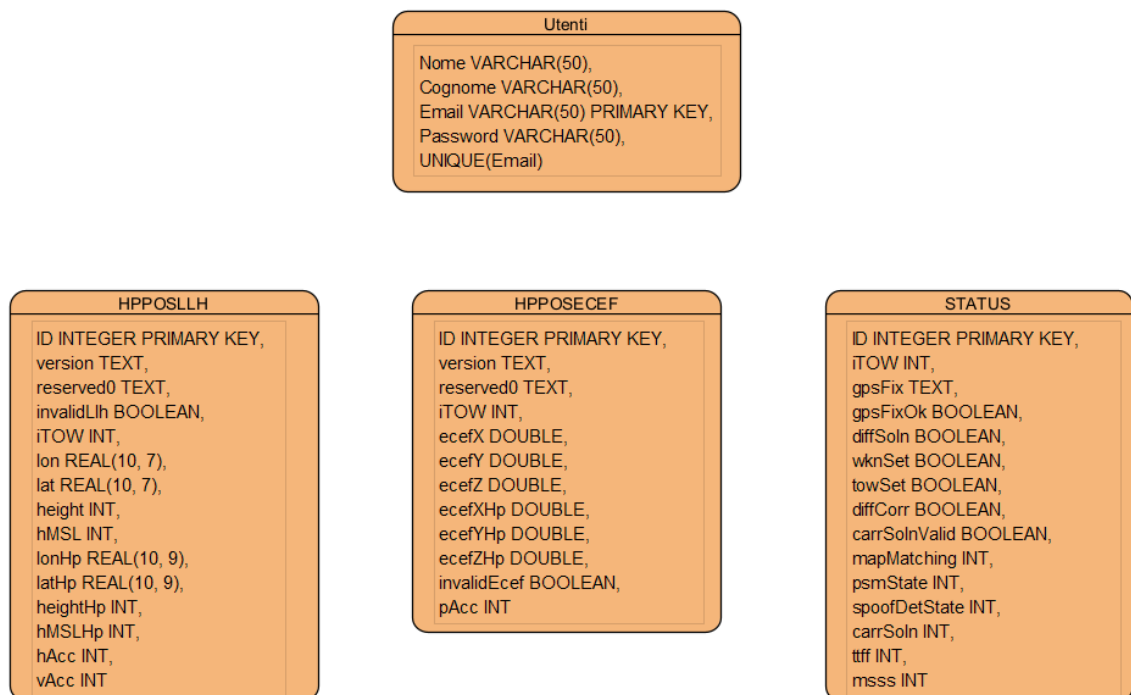


Figura 4.2. Diagramma ER

Testing

5.1 Test funzionale

Il testing è un'attività che consiste nell'eseguire il software o una sua componente con l'obiettivo di individuare difetti, errori o problemi. In particolare noi ci concentreremo sul testing funzionale, una tipologia di testing che si concentra sulla verifica delle funzionalità del software rispetto ai requisiti specificati. Nel testing funzionale, vengono progettati e eseguiti casi di test per valutare se il software funziona come previsto, eseguendo azioni specifiche e verificando i risultati ottenuti.

5.1.1 Test di unità

I test di unità sono una forma di testing che si concentra sulla verifica del corretto funzionamento delle singole unità di codice all'interno di un software. Un'unità di codice può essere una funzione, un metodo o una classe.

L'obiettivo principale dei test di unità è verificare che ogni unità di codice funzioni correttamente in modo isolato, indipendentemente dalle altre parti del software. In questo modo, si cerca di individuare eventuali difetti o errori nella logica o nell'implementazione dell'unità di codice.

Per eseguire i test di unità del sistema è stato utilizzato il framework **Pytest**. Le funzionalità testate sono:

- Login
- Registrazione
- Recupero password
- Visualizza dati

UT1 - Login

Per il testing di unità di questo requisito, si è deciso di implementare due casi di test differenti, uno in cui ci si aspetta che il test restituisca esito positivo, mentre nell'altro caso il test deve restituire esito negativo. Nel primo caso vengono passate al software le credenziali di un utente registrato in formato compatibile con quello richiesto dall'interfaccia. Nell'altro caso vengono passate al software le credenziali di un utente non registrato.

UT2 - Registrazione

Per il testing di unità della registrazione, si è deciso di implementare due casi di test con esito positivo e negativo come per il requisito precedente. Nel primo caso vengono passate al software le credenziali di un nuovo utente in formato compatibile con quello richiesto dall'interfaccia. Nel secondo caso vengono passate al software le credenziali di un utente già registrato.

UT3 - Recupero password

Per il testing di unità del requisito di recupero password, sono stati definiti due casi di test differenti per valutare i diversi scenari di esito. Nel primo caso, il test è progettato per restituire un esito positivo, mentre nel secondo caso l'aspettativa è che il test restituisca un esito negativo. Nel primo caso di test, vengono fornite al software le credenziali di un utente registrato nel formato richiesto dall'interfaccia. L'obiettivo è verificare se il software riesce correttamente a recuperare la password per un utente registrato e restituire un esito positivo. Nell'altro caso di test, vengono invece fornite al software le credenziali di un utente non registrato.

UT4 - Visualizza dati

Per il requisito di visualizzazione dei dati, sono stati implementati due casi di test. Il primo con esito positivo prevede la visualizzazione dei dati di un utente registrato che ha effettuato il login. Nel secondo caso di test, con esito negativo, un utente che non ha effettuato il login tenta la visualizzazione dei dati.

```
===== test session starts =====
platform linux -- Python 3.11.4, pytest-7.3.1, pluggy-1.0.0 -- /home/simone/anaconda3/envs/sad/bin/python
cachedir: .pytest_cache
rootdir: /home/simone/Desktop/Progetto_SAD/Codice
collected 8 items

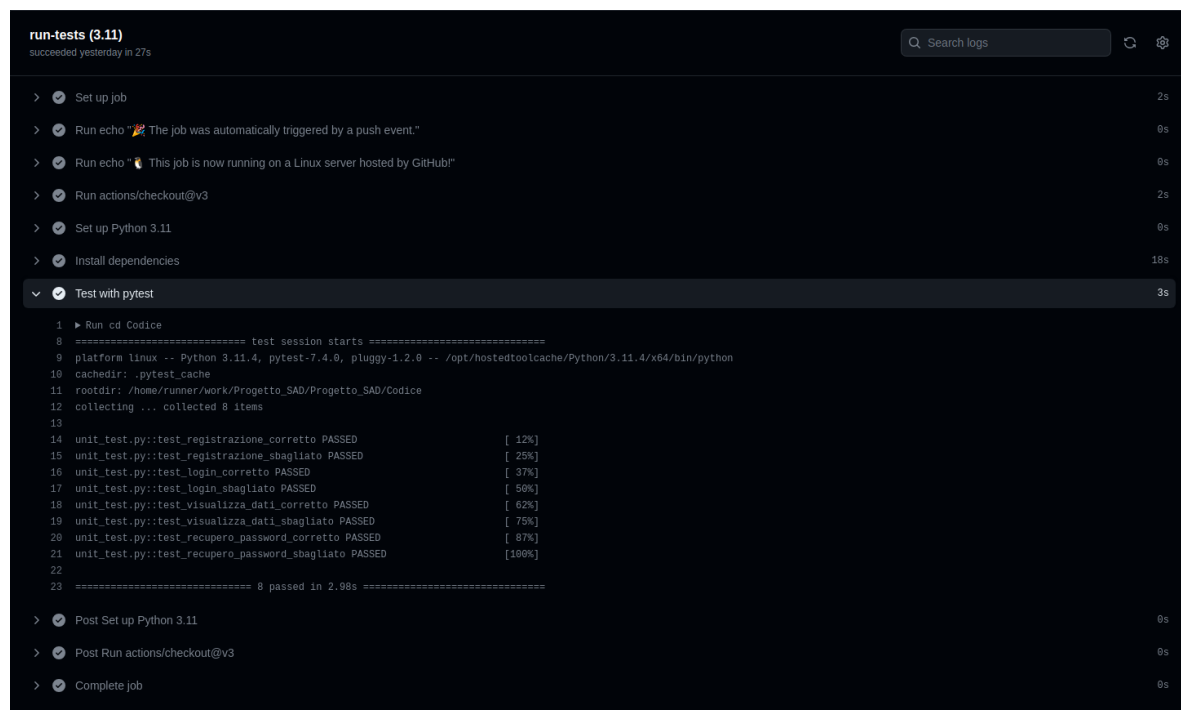
unit_test.py::test_registrazione_corretto PASSED [ 12%]
unit_test.py::test_registrazione_sbagliato PASSED [ 25%]
unit_test.py::test_login_corretto PASSED [ 37%]
unit_test.py::test_login_sbagliato PASSED [ 50%]
unit_test.py::test_recupero_password_corretto PASSED [ 62%]
unit_test.py::test_recupero_password_sbagliato PASSED [ 75%]
unit_test.py::test_visualizza_dati_corretto PASSED [ 87%]
unit_test.py::test_visualizza_dati_sbagliato PASSED [100%]

===== 8 passed in 1.17s =====
```

Figura 5.1. Test di unità

5.1.2 GitHub Actions

Le GitHub Actions sono un servizio fornito da GitHub che consente di automatizzare vari flussi di lavoro nel tuo repository GitHub. Consentono di eseguire azioni specifiche ogni volta che si verificano determinati eventi all'interno del repository, come il push di un commit, la creazione di una pull request o la pubblicazione di una release. Le GitHub Actions sono configurate tramite file YAML che definiscono i flussi di lavoro. Questi file specificano quali azioni devono essere eseguite, in quale ordine e con quali parametri. In particolare si è deciso di utilizzare le GitHub Actions per automatizzare i test. Ogni volta che viene effettuata una push sul branch "main" della repository GitHub, vengono automaticamente eseguiti i test.



```
run-tests (3.11)
succeeded yesterday in 27s

> Set up job 2s
> Run echo "The job was automatically triggered by a push event." 0s
> Run echo "This job is now running on a Linux server hosted by GitHub!" 0s
> Run actions/checkout@v3 2s
> Set up Python 3.11 0s
> Install dependencies 18s
> Test with pytest 3s

1 ▶ Run cd Codice
8 ===== test session starts =====
9 platform linux -- Python 3.11.4, pytest-7.4.0, pluggy-1.2.0 -- /opt/hostedtoolcache/Python/3.11.4/x64/bin/python
10 cachedir: .pytest_cache
11 rootdir: /home/runner/work/Progetto_SAD/Progetto_SAD/Codice
12 collecting ... collected 8 items
13
14 unit_test.py::test_registrazione_corretto PASSED [ 12%]
15 unit_test.py::test_registrazione_sbagliato PASSED [ 25%]
16 unit_test.py::test_login_corretto PASSED [ 37%]
17 unit_test.py::test_login_sbagliato PASSED [ 50%]
18 unit_test.py::test_visualizza_dati_corretto PASSED [ 62%]
19 unit_test.py::test_visualizza_dati_sbagliato PASSED [ 75%]
20 unit_test.py::test_recupero_password_corretto PASSED [ 87%]
21 unit_test.py::test_recupero_password_sbagliato PASSED [100%]
22
23 ===== 8 passed in 2.98s =====

> Post Set up Python 3.11 0s
> Post Run actions/checkout@v3 0s
> Complete job 0s
```

Figura 5.2. Test di unità

Glossario

dataframe è la collezione di acquisizioni dalla messa in funzionamento del sistema fino ad adesso.. 13

file UBX Il file UBX è un tipo di file utilizzato nei sistemi di navigazione satellitare per la comunicazione e l'interscambio di dati tra dispositivi GPS (Global Positioning System) e GNSS (Global Navigation Satellite System). UBX sta per "u-blox binary protocol" e fa riferimento al protocollo binario sviluppato dall'azienda u-blox, un produttore di tecnologie di posizionamento satellitare.. 1, 13

GPS Il GPS (Global Positioning System) è un sistema di navigazione satellitare basato su una rete di satelliti in orbita intorno alla Terra.. 1

IoT device è il dispositivo che invia il file ubx al web server.. 3

parsing Il parsing di un file è il processo di analisi e interpretazione di un file strutturato per estrarne informazioni significative.. 1

utente autenticato Si intende l'utente che ha effettuato il login al sistema.. 3

utente non autenticato Si intende l'utente che non ha effettuato il login al sistema.. 3