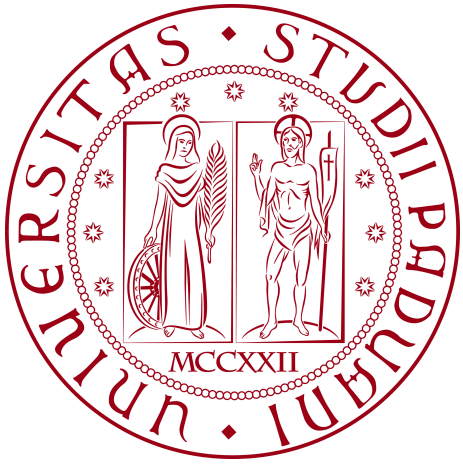
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Integrazione di Large Language Models e Data Analytics per l'Ottimizzazione delle Linee di Produzione Industriali.

Tesi di Laurea Triennale

Relatore

Prof. Zanella Marco

Laureando

Fantinato Michael

Matricola 2043672

Anno Accademico 2024-2025

Sommario

La presente tesi illustra l'ideazione, la progettazione e la prototipazione di una piattaforma software integrata in un gestionale orientata alle *Piccola e Media Impresa (PMI)*$_G$, concepita per ottimizzare i processi produttivi attraverso un'integrazione fra servizi *cloud-native*, interfacce mobile e funzionalità di raccomandazione basate su modelli di *IA*. L'elaborato percorre l'intero ciclo di vita del progetto: dall'analisi dei requisiti funzionali e non funzionali alla definizione dell'architettura, fino alla validazione sperimentale dei risultati ottenuti.

La scelta di concentrarsi su una soluzione fortemente applicativa nasce da un'esperienza maturata in contesti eterogenei: dapprima lavorando all'interno di una *PMI*$_G$ per 3 anni, quindi in una *startup* impegnata nello sviluppo di un gestionale innovativo. In entrambe le realtà è emersa con chiarezza la necessità di strumenti agili, facilmente scalabili e in grado di generare valore tangibile sui flussi di lavoro quotidiani. Collaboro con *Devess S.r.l.* da oltre un anno e, proprio durante questa esperienza, ho concepito la funzionalità di monitoraggio intelligente descritta nella tesi.

Partendo da queste considerazioni, la tesi si propone di dimostrare la concreta fattibilità tecnica del prototipo realizzato con un *back-end Python* e un *front-end* Flutter e la sua utilità potenziale nell'incrementare l'efficienza operativa. Il lavoro discute inoltre le prospettive di evoluzione futura della piattaforma e i benefici che un'adozione su larga scala potrebbe apportare all'ecosistema produttivo di un'azienda.

Acronimi e abbreviazioni

API Application Programming Interface. 1, 6, 9, 27–32

CSV Comma-Separated Values. 29, 30, 34

KPI Key Performance Indicator. 2

LLM Large Language Model. 1, 3–6, 8, 20, 27–29, 31–35

MCP Model Context Protocol. 30, 32

PMI Piccola e Media Impresa. iii, 1, 2

PoC Proof of Concept. 1, 5

UML Unified Modeling Language. 11

Glossario

agile Approccio allo sviluppo software e alla gestione dei progetti basato su cicli di lavoro iterativi e incrementali. Le metodologie agili, come Scrum, privilegiano la flessibilità, la collaborazione con il cliente e la capacità di rispondere rapidamente al cambiamento.. 26

back-end La parte di un'applicazione software che non è direttamente accessibile dall'utente finale. Comprende la logica di business, l'interazione con i database, le API e l'elaborazione dei dati. È il "motore" del sistema, in contrapposizione al *front-end*$_G$.. 27, 28, 32, 35

framework In informatica, una struttura software di base che fornisce funzionalità generiche e un'architettura predefinita. Gli sviluppatori estendono il framework con codice specifico per l'applicazione, accelerando lo sviluppo e promuovendo pratiche standard.. 27, 28, 31, 32

front-end La parte di un'applicazione software con cui l'utente interagisce direttamente, nota anche come interfaccia utente (UI). Comprende gli elementi visivi (pulsanti, menu, grafici) e la logica di presentazione dei dati.. v, 27, 35

LangChain Framework open-source per lo sviluppo di applicazioni basate su modelli linguistici (LLM). Semplifica l'orchestrazione di catene complesse (*chains*), la gestione della memoria conversazionale e l'integrazione con fonti dati e strumenti esterni.. 28, 31, 32

lead time Il termine *lead time* indica l'intervallo di tempo che intercorre tra l'avvio e il completamento di un processo produttivo o logistico, utilizzato

per misurare l'efficienza operativa e i tempi di risposta del sistema.. 1, 2, 5, 9

log In ambito informatico, il termine *log* indica un registro cronologico di eventi, messaggi o transazioni generati da un sistema, applicazione o dispositivo, utilizzato per il monitoraggio, la diagnostica e l'audit.. 15

middleware Software che agisce come intermediario tra due o più componenti applicativi, facilitandone la comunicazione e l'interoperabilità. Nel contesto di questo lavoro, si riferisce al livello logico che orchestra la comunicazione tra l'agente LLM e gli strumenti esterni.. 31

NoSQL Famiglia di sistemi di gestione di basi di dati non relazionali, progettati per offrire elevata scalabilità e flessibilità. I modelli più comuni includono database a documenti (come MongoDB), a colonne, chiave-valore e a grafo.. 33

Capitolo 1

Introduzione

Negli ultimi anni il settore manifatturiero è stato attraversato da una profonda trasformazione, nota con il paradigma dell'*Industria 4.0*, che incorpora connettività diffusa, analisi dei dati in tempo reale e sistemi intelligenti di supporto alle decisioni. In questo contesto la capacità di monitorare con precisione le performance produttive, code ai macchinari, *lead time*$_G$ e colli di bottiglia costituisce un vantaggio competitivo decisivo, soprattutto per le *PMI*$_G$ che puntano a ridurre gli sprechi e incrementare la flessibilità operativa.

La presente tesi nasce dalla collaborazione con l'azienda *Devess S.r.l.*, impegnata nello sviluppo di un gestionale innovativo. Durante un tirocinio di 300 ore è stato realizzato un *proof-of-concept* (*Proof of Concept (PoC)*$_G$) di dashboard gestionale *cross-platform*, sviluppato in Flutter e sostenuto da un back-end Python/FastAPI con database MongoDB, capace di integrare le *Application Programming Interface (API)*$_G$ di un *Large Language Model (LLM)*$_G$ per suggerire automaticamente azioni correttive o ottimizzazioni dei processi produttivi.

Per raggiungere tali traguardi è stato adottato un approccio iterativo in otto macro-fasi settimanali, dall'onboarding tecnico al rilascio del PoC, con momenti di revisione periodica con il tutor aziendale. Ciascuna fase ha previsto attività di studio, sviluppo, testing e documentazione, con un focus particolare sugli algoritmi di rilevazione delle anomalie.

Oltre a costituire un contributo tangibile all'evoluzione del gestionale di De-

vess, il lavoro di tesi dimostra come la sinergia tra Flutter, Python e LLM abiliti soluzioni agili e scalabili per la *smart manufacturing*, favorendo decisioni proattive e riducendo il *time-to-action* dei responsabili di produzione.

1.1 L'azienda

Fondata nel 2020, *Devess S.r.l.* ha sede in Via Per Curnasco 58, Bergamo, e si occupa di consulenza e realizzazione di software gestionali per il settore manifatturiero. La piattaforma proprietaria *Operativo.io* consente alle aziende di monitorare in tempo reale il flusso produttivo, analizzare i *Key Performance Indicators* (*Key Performance Indicator (KPI)*$_G$) e intervenire prontamente in caso di inefficienze; proprio in questo software verrà implementata la nuova dashboard.

La mission di Devess è migliorare la gestione degli ordini nelle *PMI*$_G$, coniugando l'esperienza dei processi produttivi tradizionali con le potenzialità offerte dal digitale e l'industria 4.0. L'orientamento alla *data-driven decision making* sta rendendo l'azienda una valida soluzione per le *PMI*$_G$ che desiderano intraprendere un percorso di trasformazione digitale, contenendo tempi e costi di adozione.

Durante il tirocinio il tutor aziendale, Nicola Romano, ha coordinato incontri settimanali — sia in presenza sia in modalità telematica — per monitorare l'avanzamento, fornire feedback puntuali e garantire l'allineamento agli obiettivi di progetto O01–O05 definiti nel piano di lavoro.

1.2 L'idea

Il progetto di stage si proponeva di sviluppare un sistema dimostrativo che integrasse analisi dati, visualizzazione interattiva e suggerimenti generati da un modello linguistico di grandi dimensioni. L'idea era duplice:

1. **Visualizzazione delle criticità**. Realizzare una dashboard che mettesse in evidenza code alle macchine, *lead time*$_G$ e colli di bottiglia trovati tramite l'analisi dei dati presenti nel database MongoDB.

2. **Supporto decisionale intelligente**. Sfruttare un LLM_G per suggerire, alla luce dei dati analizzati, azioni correttive concrete con potenziale estensione a un chatbot.

La combinazione di queste componenti permette al management di individuare una criticità in pochi clic e di ricevere, contestualmente, soluzioni suggerite dall'intelligenza artificiale. Tale approccio, oltre a ridurre i tempi di analisi, favorisce una cultura operativa basata su *insight* immediati e misurabili.

1.3 Organizzazione del testo

Il secondo capitolo documenta l'impostazione operativa dello stage, le modalità di interazione con l'azienda e con il tutor, l'approccio metodologico adottato e l'analisi dei principali rischi.

Il terzo capitolo presenta un'esame approfondito dei requisiti, suddivisi in obbligatori, desiderabili e opzionali, corredato dalla mappatura di attori, *stakeholder*, *use case* e funzionalità previste.

Il quarto capitolo espone i fondamenti teorici e tecnologici alla base del progetto, confronta le soluzioni esistenti e argomenta la scelta degli strumenti impiegati.

Il quinto (ed eventuali sesto e settimo) capitolo descrive in dettaglio le attività eseguite, le criticità incontrate e le soluzioni adottate, includendo schemi, tabelle e immagini ove opportuno.

Il capitolo conclusivo raccoglie le riflessioni finali: valutazione del raggiungimento degli obiettivi, competenze acquisite, possibili sviluppi futuri e considerazioni personali.

Riguardo alla stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;

- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *LLM*$_G$;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Descrizione dello stage

2.1 Introduzione al progetto

Negli ultimi anni il settore manifatturiero ha intrapreso un percorso di digitalizzazione incentrato sul paradigma dell'*Industria 4.0*. In tale contesto il presente tirocinio, svolto presso l'azienda *Devess S.r.l.*, si propone di realizzare un *proof-of-concept* (*PoC*$_G$) da implementare in un software gestionale *cross-platform* sviluppato con Flutter, supportato da un back-end *Python*/FastAPI con persistenza in MongoDB e integrazione di un modello linguistico di grandi dimensioni (*LLM*$_G$). L'obiettivo primario è consentire al management di individuare tempestivamente criticità operative, code ai macchinari, *lead time*$_G$, anomalie, colli di bottiglia e di ricevere suggerimenti automatici di ottimizzazione basati sui dati.

Il progetto si articola in otto macro-fasi (cfr. sezione 2.4), per un totale di 300 ore, e combina attività di analisi dati, *prompt engineering*, sviluppo *mobile* e validazione sperimentale. Il tutor aziendale, Nicola Romano, ha seguito l'avanzamento mediante allineamenti settimanali sui risultati ottenuti, facilitando l'accesso alle risorse necessarie.

2.2 Analisi preventiva dei rischi

1. Qualità dei dati in MongoDB

Descrizione: I dati storici potrebbero presentare valori mancanti o incoerenti, compromettendo l'accuratezza delle analisi e dei suggerimenti generati dall'intelligenza artificiale.

Soluzione: Definire procedure di *data cleaning* automatizzate e introdurre controlli di validazione a campione.

2. Costi e limiti di utilizzo delle *API*$_G$ LLM

Descrizione: L'uso intensivo delle *API*$_G$ di ChatGPT potrebbe superare il budget assegnato o incorrere in *rate-limit*, rendendo la funzionalità economicamente insostenibile.

Soluzione: Svolgere un'accurata analisi dei costi e monitorare i consumi del prototipo tramite *dashboard* dedicata, al fine di stimare il costo medio d'utilizzo e adeguare di conseguenza il pricing del software.

3. non-determinismo delle risposte da parte del LLM

Descrizione: La natura probabilistica del *LLM*$_G$ può produrre risposte diverse a parità di input, introducendo possibili incoerenze o inesattezze nei suggerimenti operativi.

Soluzione: Utilizzare istruzioni precise tramite *prompt engineering*, e prevedere un *workflow* di approvazione manuale per le decisioni critiche.

4. Scalabilità del back-end

Descrizione: L'aumento del volume di dati o di richieste simultanee potrebbe ridurre le prestazioni degli endpoint FastAPI.

Soluzione: Progettare l'architettura con attenzione alla scalabilità, selezionare il deployment più adatto a un utilizzo su larga scala, adottare containerizzazione e bilanciamento del carico, prevedendo la replica del database e test di *stress*.

5. Usabilità dell'interfaccia Flutter

Descrizione: Un'interfaccia poco intuitiva potrebbe vanificare i benefici dell'automazione, rallentando l'adozione da parte del management.

Soluzione: Applicare le linee guida Material 3, condurre *usability test* con utenti interni e iterare sulle funzionalità maggiormente critiche.

2.3 Requisiti e obiettivi

Codice	Descrizione
O01	Dashboard Flutter che visualizzi dati di produzione, anomalie e metriche operative in tempo reale, integrata nel gestionale mobile di Devess.
O02	Integrazione di GPT come LLM_G per restituire risposte e suggerimenti in linguaggio naturale basati sui dati.
O03	Implementazione di funzioni analitiche in grado di identificare anomalie nei processi produttivi.
O04	Sistema di autenticazione sicura con controllo degli accessi.
O05	Interfaccia utente intuitiva con flussi di navigazione semplificati per operatori di produzione.
D01	Realizzazione di una versione *web* dell'applicazione, integrata nel gestionale Devess.
D02	Possibilità di interagire con un chatbot basato su LLM_G per richiedere chiarimenti o soluzioni mirate.
D03	Sistema di notifiche push o e-mail proattive al manifestarsi di problemi o rallentamenti nella produzione.
F01	Sincronizzazione offline con riallineamento automatico al ripristino della connettività.
F02	Animazioni fluide e tempi di caricamento percepiti veloci.
F03	Inserimento comandi in linguaggio naturale.

Tabella 2.2: Requisiti e obiettivi definiti per il tirocinio.

2.4 Pianificazione

Il lavoro è suddiviso in otto settimane; ciascuna prevede attività specifiche, *milestone* e momenti di verifica con il tutor aziendale.

1. **Settimana 1 — Onboarding e configurazione ambienti (40 h)**

- Configurazione degli ambienti *Python*/FastAPI, Flutter e MongoDB.
- Studio delle *API*$_G$ di ChatGPT e definizione degli *use case* AI.

2. **Settimana 2 — Prompt engineering e integrazione base LLM (40 h)**
   - Applicazione di tecniche di *prompt engineering*.
   - Prima integrazione delle *API*$_G$ LLM con tracciamento di costi e latenza.
3. **Settimana 3 — Modellazione dati e MongoDB (40 h)**
   - Definizione dello schema NoSQL per macchinari e produzione.
   - Importazione e indicizzazione dei dati aziendali, con politiche di sicurezza.
4. **Settimana 4 — Back-end Python e algoritmi analitici (40 h)**
   - Realizzazione degli endpoint CRUD.
   - Sviluppo degli algoritmi per code, *lead time*$_G$ e colli di bottiglia.
5. **Settimana 5 — Validazione algoritmi e suggerimenti AI (40 h)**
   - Valutazione del corretto funzionamento degli algoritmi di analisi dati.
   - Integrazione con ChatGPT per suggerimenti contestuali.
6. **Settimana 6 — Automazione e performance (40 h)**
   - Scheduler per analisi periodiche.
   - Ottimizzazione delle query e monitoraggio dei costi operativi.
7. **Settimana 7 — UI e dashboard Flutter (40 h)**
   - Progettazione del *design system* secondo Material 3.
   - Implementazione degli schermi di monitoraggio e test multi-device.
8. **Settimana 8 — QA, MLOps e rilascio (20 h)**

- Test end-to-end e preparazione del pacchetto di rilascio.
- Stesura della documentazione tecnica conclusiva.

2.4.1 Modalità di interazione con il tutor aziendale

Sono previsti incontri di avanzamento con il tutor ogni settimana, in presenza o da remoto, per discutere le criticità emerse, verificare il rispetto del piano e concordare eventuali adeguamenti.

2.4.2 Deliverable previsti

- **PoC** della dashboard gestionale completa di analisi e suggerimenti AI.
- Report tecnico finale strutturato nei capitoli di introduzione, architettura, metodologie, risultati e conclusioni.
- Codice sorgente versionato su repository Git proprietaria.

Capitolo 3

Analisi dei requisiti

3.1 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso sono diagrammi di tipo *Unified Modeling Language (UML)*$_G$ dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso. Di seguito i casi d'uso definitivi, suddivisi per priorità.

3.1.1 Casi d'uso obbligatori

I seguenti casi d'uso rappresentano le funzionalità fondamentali che il sistema deve necessariamente implementare per soddisfare i requisiti minimi del progetto.

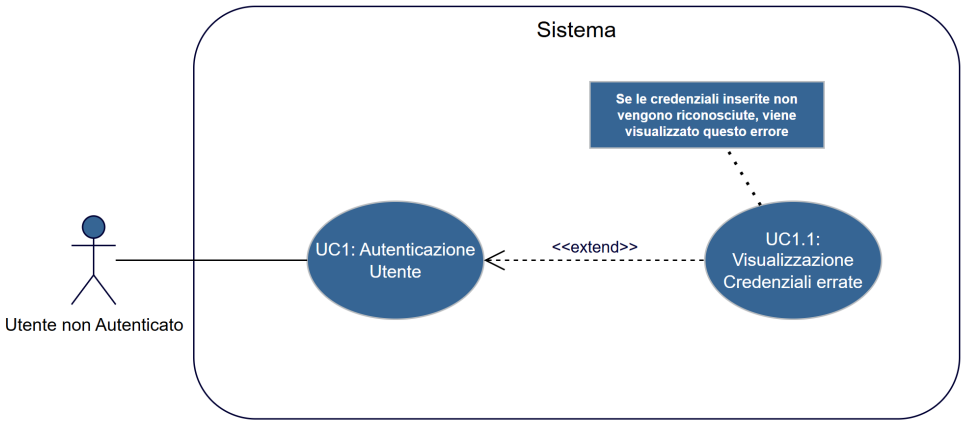


Figura 3.1: UC1 + UC1.1.

UC1: Autenticazione Utente

Attori Principali: Utente non autenticato

Precondizioni: L'utente si trova alla schermata di login.

Descrizione: Il Sistema verifica le credenziali inserite e, se corrette, crea una sessione autenticata protetta.

Postcondizioni: L'utente accede alle funzionalità.

Scenario Alternativo: Se le credenziali non sono valide, viene mostrato un messaggio di errore (UC1.1).

UC1.1: Visualizzazione Credenziali Errate

Attori Principali: Utente non autenticato

Precondizioni: L'utente si trova alla schermata di login.

Descrizione: Il Sistema verifica le credenziali inserite e, se scorrette, mostra un messaggio di errore.

Postcondizioni: L'utente può riprovare reinserendo le credenziali.

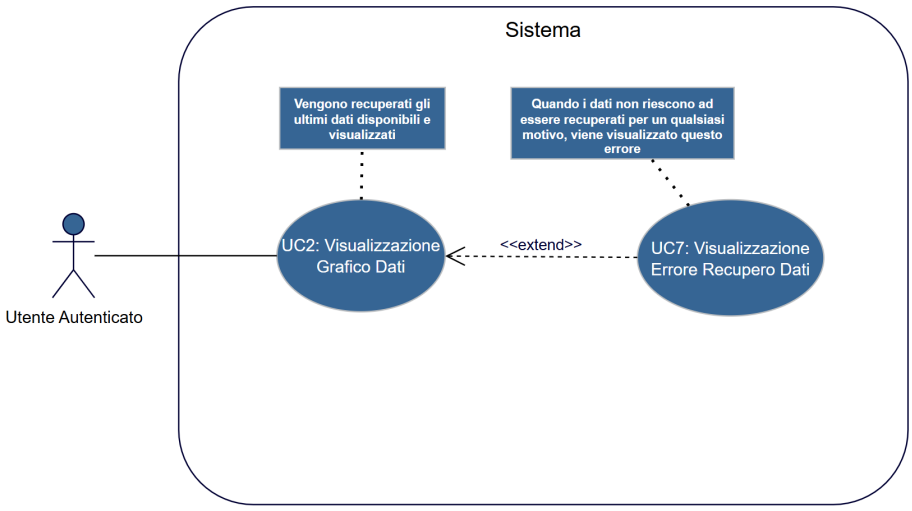


Figura 3.2: UC2 + UC7.

UC2: Visualizzazione Grafico Dati

Attori Principali: Utente

Precondizioni: L'utente è autenticato.

Descrizione: Il Sistema mostra un grafico con l'andamento temporale dei dati di produzione.

Postcondizioni: Il grafico viene aggiornato con l'ultima sincronizzazione disponibile.

Scenario Alternativo: Se i dati non sono presenti, viene visualizzato un messaggio di errore (UC7).

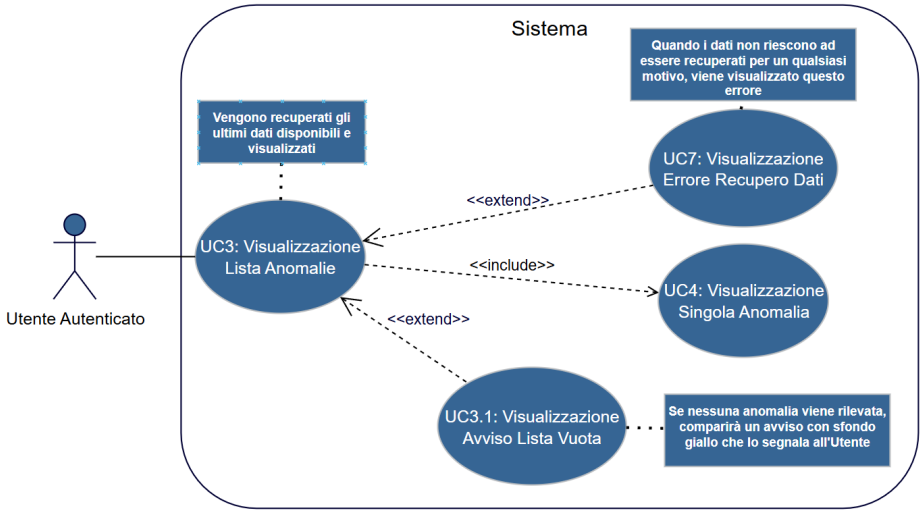


Figura 3.3: UC3 + UC4 + UC7.

UC3: Visualizzazione Lista Anomalie

Attori Principali: Utente

Precondizioni: Sono state eseguite analisi sui dati.

Descrizione: Il Sistema elenca tutte le anomalie rilevate.

Postcondizioni: La lista è disponibile per ulteriori approfondimenti.

Scenario Alternativo: Se non esistono anomalie nel periodo selezionato, viene mostrato un messaggio informativo (UC3.1).

Scenario Alternativo: Se il sistema non riesce a recuperare i dati, viene visualizzato un errore (UC7).

UC3.1: Visualizzazione Avviso Lista Vuota

Attori Principali: Utente

Precondizioni: Il Sistema notifica che non sono state rilevate anomalie.

Descrizione: Viene visualizzato un avviso indicando l'assenza di anomalie.

UC4: Dettaglio Anomalia

Attori Principali: Utente

Precondizioni: L'utente ha selezionato una specifica anomalia dalla lista.

Descrizione: Il Sistema visualizza grafici e log_G di dettaglio relativi all'anomalia scelta.

Postcondizioni: L'utente può valutare la necessità di un intervento correttivo.

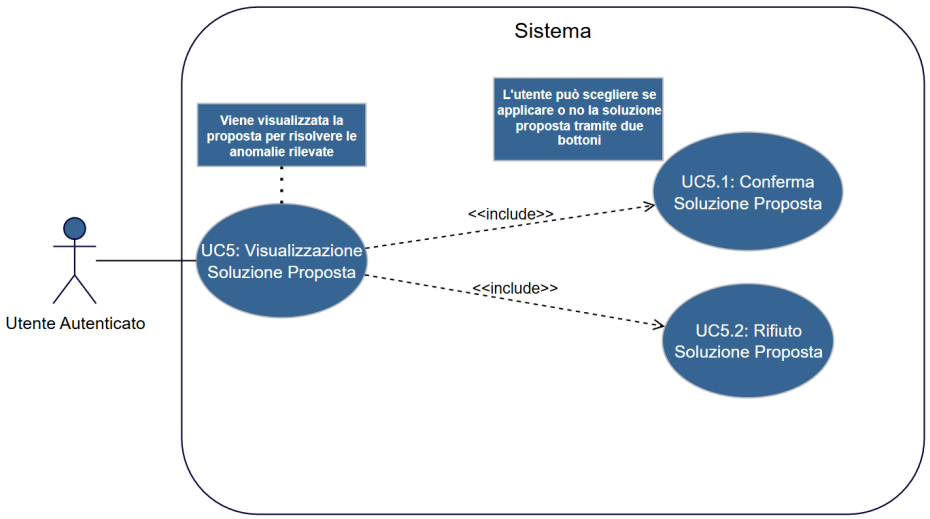


Figura 3.4: UC5 + UC5.1 + UC5.2.

UC5: Visualizzazione Soluzione Proposta

Attori Principali: Utente

Precondizioni: È stato selezionato un evento critico.

Descrizione: Il Sistema visualizza una proposta di soluzione.

Postcondizioni: La soluzione può essere accettata (UC5.2) o ignorata (UC5.1).

UC5.1: Rifiuto Soluzione Proposta

Attori Principali: Utente

Precondizioni: È visualizzata una soluzione proposta (UC5).

Descrizione: L'Utente rifiuta la soluzione proposta.

Postcondizioni: Il sistema registra il rifiuto e mantiene l'anomalia aperta.

UC5.2: Conferma Soluzione Proposta

Attori Principali: Utente

Precondizioni: È visualizzata una soluzione proposta (UC5).

Descrizione: L'Utente approva la soluzione che viene applicata al sistema di produzione.

Postcondizioni: Il sistema avvia l'esecuzione del piano correttivo.

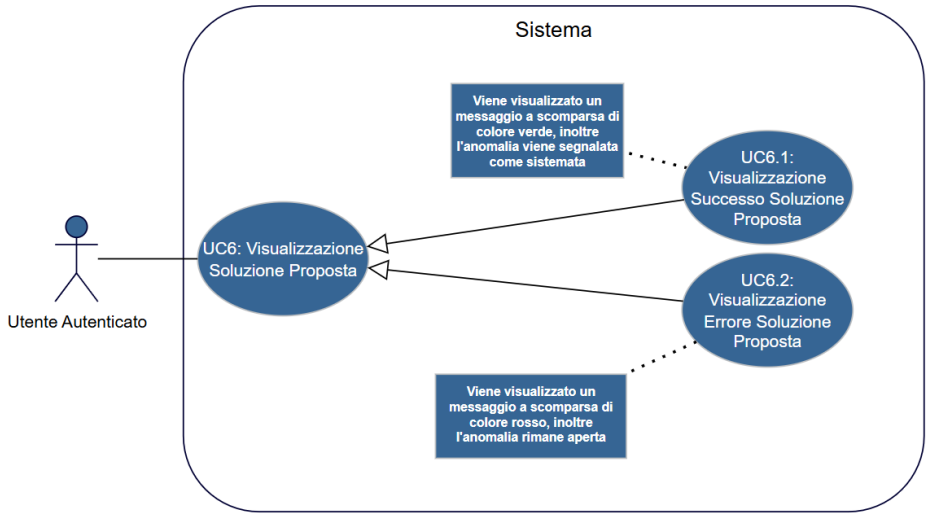


Figura 3.5: UC6 + UC6.1 + UC6.2.

UC6: Visualizzazione Risultato Soluzione Proposta

Attori Principali: Utente

Precondizioni: Il piano correttivo è stato approvato ed eseguito.

Descrizione: Il Sistema notifica all'Utente il risultato.

Postcondizioni: In base all'esito dell'applicazione, il messaggio può essere positivo (UC6.1) o negativo (UC6.2).

UC6.1: Visualizzazione Successo Soluzione Proposta

Attori Principali: Utente

Precondizioni: Il piano correttivo è stato eseguito correttamente.

Descrizione: Il Sistema notifica all'Utente l'avvenuta applicazione della soluzione.

Postcondizioni: L'anomalia viene marcata come risolta.

UC6.2: Visualizzazione Errore Soluzione Proposta

Attori Principali: Utente

Precondizioni: Il sistema ha tentato di applicare una soluzione proposta.

Descrizione: Il Sistema segnala all'Utente il fallimento dell'applicazione e fornisce i dettagli dell'errore.

Postcondizioni: L'anomalia resta aperta e viene registrato il tentativo fallito.

UC7: Visualizzazione Errore Recupero Dati

Attori Principali: Utente

Precondizioni: L'utente è autenticato.

Descrizione: Il Sistema mostra un errore perché i dati non sono presenti.

Postcondizioni: L'Utente può aggiornare la pagina per riprovare a recuperare i dati.

3.1.2 Casi d'uso desiderabili

I seguenti casi d'uso rappresentano le funzionalità desiderabili che il sistema potrebbe implementare per migliorare l'esperienza dell'utente e fornire funzionalità aggiuntive.

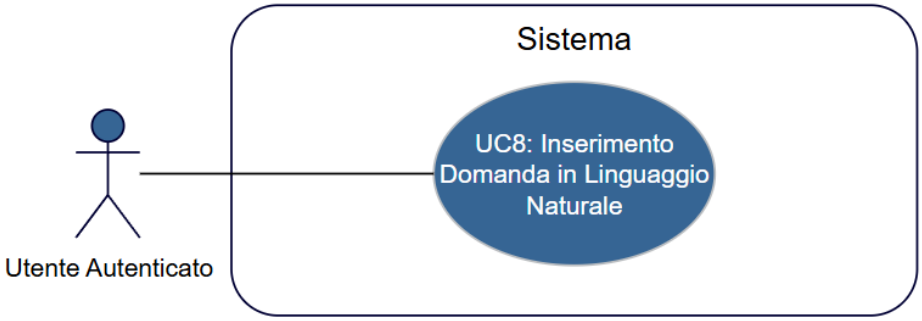


Figura 3.6: UC8.

UC8: Inserimento Domanda in Linguaggio Naturale

Attori Principali: Utente

Precondizioni: L'utente è autenticato.

Descrizione: L'utente inserisce una domanda libera relativa al sistema.

Postcondizioni: La domanda viene inoltrata al LLM_G.

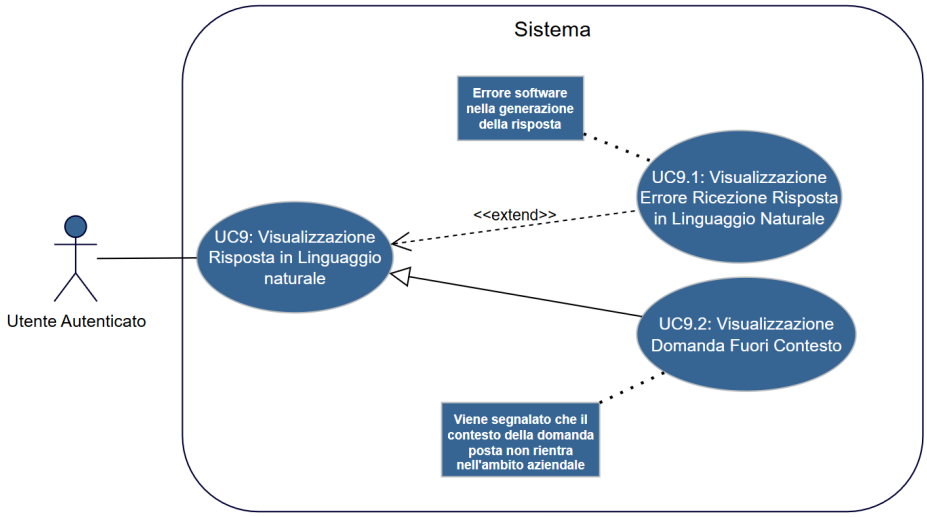


Figura 3.7: UC9 + UC9.1 + UC9.2.

UC9: Visualizzazione Risposta in Linguaggio Naturale

Attori Principali: Utente

Precondizioni: È stata inoltrata una domanda (UC8).

Descrizione: Viene visualizzata una risposta generata in linguaggio naturale.

Postcondizioni: La risposta è visualizzata all'Utente.

Scenario Alternativo: Se si verifica un errore, viene mostrato un messaggio di errore (UC9.1).

Scenario Alternativo: Se la domanda è fuori contesto, il Sistema chiede all'utente di riformulare (UC9.2).

UC9.1: Visualizzazione Errore Ricezione Risposta in Linguaggio Naturale

Attori Principali: Utente

Precondizioni: È stata inoltrata una domanda (UC8).

Descrizione: Il sistema non riesce a recuperare la risposta.

Postcondizioni: L'utente visualizza un errore che lo informa del problema.

UC9.2: Visualizzazione domanda fuori contesto

Attori Principali: Utente

Precondizioni: È stata inoltrata una domanda (UC8) non inerente all'ambito operativo.

Descrizione: L'utente visualizza un avviso che la domanda posta era fuori contesto.

Postcondizioni: L'utente può riformulare la domanda.

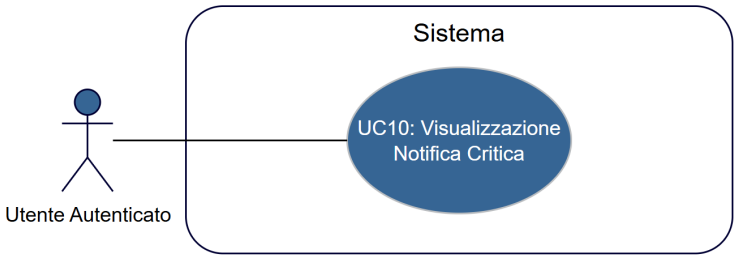


Figura 3.8: UC10.

UC10: Visualizzazione Notifica Critica

Attori Principali: Utente

Precondizioni: Il sistema rileva un'anomalia critica.

Descrizione: Viene visualizzato un avviso sui dispositivi connessi.

Postcondizioni: Gli operatori visualizzano l'avviso.

3.1.3 Casi d'uso facoltativi

I seguenti casi d'uso rappresentano le funzionalità facoltative che il sistema potrebbe implementare per migliorare l'esperienza dell'utente e fornire funzionalità aggiuntive.

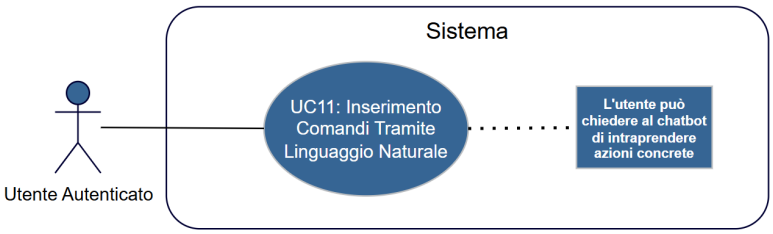


Figura 3.9: UC11.

UC11: Inserimento Comandi Tramite Linguaggio Naturale

Attori Principali: Utente

Precondizioni: L'utente è autenticato e l'interfaccia chat è attiva.

Descrizione: L'utente impartisce comandi operativi in linguaggio naturale; il Sistema li interpreta e li esegue.

Postcondizioni: L'azione richiesta viene completata o ne viene comunicato l'esito.

3.2 Tracciamento dei requisiti

Il tracciamento mostra la copertura dei requisiti funzionali (F) classificati come **N** (obbligatori), **D** (desiderabili) e **Z** (facoltativi).

Requisito	Descrizione	Use Case
FN1	Login sicuro dell'utente	UC1, UC1.1
FN2	Visualizzare grafico dei dati di produzione	UC2, UC7
FN3	*Dashboard* anomalie rilevate	UC3, UC3.1, UC7, UC4
FN4	Presentare una soluzione proposta tramite LLM	UC5, UC5.1, UC5.2
FN5	Gestire l'accettazione/rifiuto di una soluzione	UC5.1, UC5.2
FN6	Notificare risultato applicazione soluzione	UC6, UC6.1, UC6.2
FN7	Gestire errori di recupero dati	UC7
FD1	Inserire domande in linguaggio naturale	UC8
FD2	Ricevere risposte in linguaggio naturale	UC9, UC9.1, UC9.2
FD3	Visualizzare notifiche critiche	UC10
FZ1	Inserire comandi in linguaggio naturale	UC11

Tabella 3.2: Tracciamento dei requisiti funzionali.

Capitolo 4

Analisi Comparativa delle Tecnologie e degli Strumenti

Lo sviluppo di un sistema software innovativo, specialmente in un contesto *agile*$_G$ come quello di una startup, richiede un'attenta valutazione delle tecnologie disponibili. Ogni scelta architetturale implica un compromesso tra *performance*, costi, tempi di sviluppo e manutenibilità futura. Questo capitolo non si limita a elencare gli strumenti adottati per la realizzazione del prototipo, ma si propone di analizzare criticamente il panorama tecnologico per ciascun dominio problematico affrontato.

L'analisi seguirà una struttura sistematica per ogni componente dell'architettura:

1. **Definizione del problema**: si delineerà la sfida tecnica o funzionale da risolvere.

2. **Analisi delle alternative**: si esamineranno le principali tecnologie e approcci disponibili sul mercato per affrontare tale sfida, valutandone pro e contro.

3. **Soluzione adottata e motivazioni**: si presenterà la scelta effettuata, giustificandola non solo in base a meriti tecnici intrinseci, ma anche in relazione al contesto specifico del progetto, ovvero l'ecosistema di Devess, gli obiettivi della tesi e i vincoli operativi.

4.1 Architettura del Back-end e Comunicazione API

4.1.1 Definizione del problema

Il *back-end*$_G$ rappresenta il nucleo logico dell'applicazione. La sfida consiste nel creare un servizio che abbia la duplice responsabilità di esporre un'interfaccia *API*$_G$ sicura e performante per il *front-end*$_G$ e di orchestrare le complesse interazioni con i modelli linguistici e le fonti dati esterne. La sua progettazione è cruciale per la reattività e la scalabilità dell'intero sistema, specialmente dovendo gestire chiamate asincrone a lunga attesa verso servizi di terze parti come gli *LLM*$_G$.

4.1.2 Analisi delle alternative

La scelta del *framework*$_G$ e del linguaggio di programmazione per il *back-end*$_G$ ha considerato tre principali candidati:

- **Python con Django/Flask**: Python è il linguaggio d'elezione per il *machine learning* e l'analisi dati. **Django** è un *framework*$_G$ "*batteries-included*", robusto e maturo, ideale per progetti complessi con requisiti ben definiti. **Flask** è un *micro-framework* minimale e flessibile, che lascia massima libertà allo sviluppatore. Tuttavia, entrambi richiedono configurazioni aggiuntive per una gestione efficiente della concorrenza asincrona, fondamentale per il nostro caso d'uso.

- **Node.js con Express.js/NestJS**: L'ecosistema JavaScript/TypeScript con Node.js è rinomato per il suo modello di *I/O non bloccante*, che lo rende eccezionalmente performante per applicazioni *real-time* e *API-intensive*. **Express.js** è il *framework*$_G$ minimale di riferimento, mentre **NestJS** offre una struttura più opinionata e scalabile, ispirata ad Angular. Sebbene performante, l'ecosistema di Node.js per l'analisi dati e

l'interazione scientifica non eguaglia ancora la maturità e la ricchezza di quello Python.

- **Python con FastAPI**: **FastAPI** è un *framework*$_G$ moderno che unisce la semplicità di Flask con *performance* paragonabili a quelle di Node.js, grazie al supporto nativo per la programmazione asincrona (tramite ASGI). Offre inoltre vantaggi unici come la validazione dei dati basata sui *type hint* di Python tramite Pydantic e la generazione automatica di documentazione interattiva (*Swagger UI*), accelerando drasticamente il ciclo di sviluppo e test delle *API*$_G$.

4.1.3 Soluzione adottata e motivazioni

La scelta è ricaduta su **Python** e sul *framework*$_G$ **FastAPI**. Le motivazioni sono strategiche e tecniche:

1. **Ecosistema Python**: La necessità di integrare librerie di analisi dati come Pandas e di interagire fluidamente con *framework*$_G$ di orchestrazione *LLM*$_G$ (come *LangChain*$_G$) ha reso Python la scelta più naturale e produttiva.

2. ***Performance* Asincrone**: FastAPI gestisce nativamente le operazioni asincrone. Questo è un requisito non negoziabile, in quanto il *backend*$_G$ deve attendere le risposte dall'*LLM*$_G$ senza bloccare altre richieste concorrenti.

3. **Produttività dello Sviluppatore**: La validazione automatica di Pydantic e la documentazione OpenAPI riducono il codice ripetitivo e il rischio di errori.

4. **Coerenza Aziendale**: Python è già ampiamente utilizzato nell'azienda, il che rende la manutenibilità futura più facile e immediata, allineandosi alle competenze interne.

4.2 Selezione del Modello Linguistico

4.2.1 Definizione del problema

La scelta del *LLM*$_G$ è il fulcro del sistema intelligente. Il problema non è selezionare un modello qualsiasi, ma individuare quello le cui capacità si allineano meglio ai requisiti specifici del progetto: non solo comprendere e generare linguaggio naturale, ma anche seguire istruzioni complesse, invocare strumenti esterni in modo affidabile e, soprattutto, analizzare dati strutturati forniti in formati comuni come il *Comma-Separated Values (CSV)*$_G$. La scelta deve bilanciare *performance* di ragionamento, costi operativi e facilità di integrazione.

4.2.2 Analisi delle alternative

Sono stati presi in considerazione i principali attori del mercato dei modelli linguistici:

- **Modelli OpenAI (famiglia GPT)**: Rappresentano lo stato dell'arte per molteplici compiti di ragionamento generale. La loro *API*$_G$, con la funzionalità di *Function Calling*, è matura e ben documentata. Tuttavia, la loro capacità di analizzare direttamente file di dati strutturati può richiedere passaggi intermedi di pre-elaborazione.
- **Modelli Google (famiglia Gemini)**: Sono potenti alternative multimodali con un forte ecosistema alle spalle. Al momento della valutazione, le loro capacità di *tool-calling* erano in rapida evoluzione ma potenzialmente meno consolidate rispetto ai concorrenti diretti per questo specifico caso d'uso.
- **Modelli Open-Source (es. Llama, Mixtral)**: Offrono il vantaggio del pieno controllo e dell'assenza di costi per *API*$_G$, consentendo anche un *fine-tuning* spinto. Di contro, richiedono un notevole onere infrastrutturale per l'*hosting*, la gestione e la scalabilità. Inoltre, le loro capacità di seguire catene di ragionamento complesse e di usare strumenti in mo-

do affidabile, sebbene in costante miglioramento, spesso non raggiungono ancora la robustezza dei migliori modelli proprietari.

4.2.3 Soluzione adottata e motivazioni

È stato scelto un modello della famiglia **Claude 4 di Anthropic**. Le ragioni di questa decisione sono strettamente legate ai requisiti funzionali del prototipo:

- **Capacità di Analisi Dati**: I modelli Claude hanno dimostrato una capacità superiore nell'analizzare e interpretare direttamente il contenuto di file con dati strutturati, come il *CSV*$_G$, questa abilità è fondamentale per il nostro utilizzo.
- **Compatibilità con il Protocollo *Model Context Protocol (MCP)*$_G$**: Il meccanismo nativo di *tool-use* di Claude è stato una forte ispirazione per la progettazione del *MCP*$_G$. Di conseguenza, il protocollo custom sviluppato ha una compatibilità quasi nativa con il modo in cui Claude gestisce l'invocazione di strumenti esterni. Questo ha ridotto al minimo l'attrito implementativo, garantendo un'integrazione fluida e affidabile.
- **Equilibrio Costo/Performance**: I modelli della famiglia Claude 4, in particolare la versione Sonnet, offrono un eccellente equilibrio tra capacità di ragionamento di alto livello e costi di utilizzo dell'*API*$_G$, rendendoli una scelta sostenibile per un prototipo destinato a evolversi.

4.3 Orchestrazione dell'Agente Intelligente

4.3.1 Definizione del problema

Il cuore innovativo della tesi risiede nella trasformazione dell'*LLM*$_G$ in un *agente* autonomo. La sfida non è solo quella di generare risposte testuali, ma di creare un sistema in cui l'*LLM*$_G$ possa eseguire azioni concrete invocando strumenti esterni. È necessario un *middleware*$_G$ capace di orchestrare il dialogo, gestire la memoria conversazionale e permettere al modello di interagire con il mondo esterno in modo sicuro, affidabile e strutturato.

4.3.2 Analisi delle alternative

Per l'orchestrazione dell'*agente* sono stati valutati tre approcci principali:

- **Sviluppo *Custom* senza Framework**: L'approccio più basilare consiste nell'interagire direttamente con le *API*$_G$ del *provider* *LLM*$_G$ (es. OpenAI, Anthropic). Sebbene offra il massimo controllo, richiede la re-implementazione di logiche complesse e soggette a errori: gestione dello storico, *parsing* delle risposte, gestione dei tentativi e degli errori. Il carico di lavoro in termini di codice *boilerplate* è proibitivo per un prototipo.

- **LlamaIndex**: È un *framework*$_G$ eccellente, specializzato nel *Retrieval-Augmented Generation* (RAG). Il suo punto di forza è la costruzione e l'interrogazione di indici su grandi volumi di dati. Sebbene potente, il suo *focus* è primariamente sull'arricchimento del contesto tramite recupero di informazioni. Il nostro caso d'uso, che richiede l'esecuzione di azioni tramite strumenti eterogenei, necessita di un'astrazione più generica.

- ***LangChain***$_G$: Si posiziona come un *framework*$_G$ *general-purpose* per la costruzione di applicazioni basate su *LLM*$_G$. Offre astrazioni di alto livello per concetti come *chains*, memoria, *agenti* e strumenti. La sua architettura modulare semplifica enormemente la definizione di *agenti* complessi e l'integrazione di *tool custom*, permettendo di definire la logica in modo dichiarativo.

4.3.3 Soluzione adottata e motivazioni

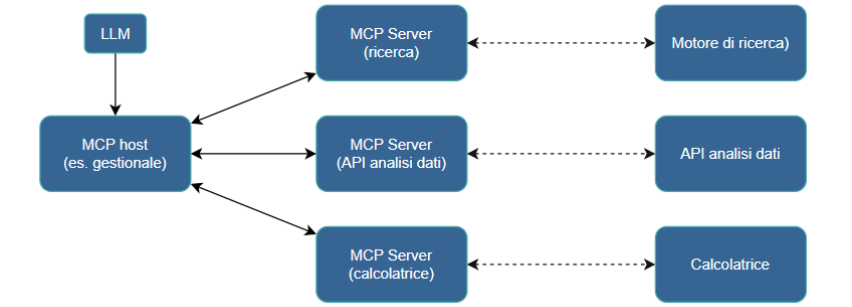


Figura 4.1: Schema di funzionamento MCP

La soluzione adottata si basa sulla sinergia tra ***LangChain***$_G$ e un protocollo custom, il ***MCP***$_G$. *LangChain*$_G$ è stato scelto come *framework*$_G$ di orchestrazione perché permette di astrarre la complessità della comunicazione con l'*LLM*$_G$, fornendo blocchi pre-costruiti per la gestione della memoria e per l'implementazione del ciclo Ragione-Azione (*ReAct*).

A complemento, il *MCP*$_G$ è stato definito per disaccoppiare la logica degli strumenti dal modello specifico. Mentre le *API*$_G$ native (es. OpenAI Function Calling) legano l'implementazione a un singolo *provider*, il *MCP*$_G$ definisce un contratto agnostico. Questa scelta garantisce:

- **Flessibilità**: Possibilità di sostituire l'*LLM*$_G$ sottostante senza riscrivere la logica degli strumenti.
- **Controllo e Sicurezza**: Centralizzazione della validazione e dell'esecuzione degli strumenti sul nostro *back-end*$_G$.
- **Estensibilità**: Semplificazione nell'aggiunta di nuovi strumenti.

4.4 Gestione e Persistenza dei Dati

4.4.1 Definizione del problema

La gestione dei dati si articola su due sfide distinte: la persistenza affidabile dei dati operativi dell'applicazione (sessioni utente, conversazioni) e la gestione dei *set* di dati derivati dall'analisi, che devono essere archiviati e resi disponibili per l'uso da parte dell'LLM_G. Le scelte in questo ambito dovevano tenere conto dell'infrastruttura tecnologica preesistente in azienda per minimizzare l'impatto operativo.

4.4.2 Analisi delle alternative

L'analisi si è concentrata su due domini: il database primario e il formato di esportazione dati.

Alternative per il Database Primario

Un *database* relazionale (SQL) come **PostgreSQL** rappresentava la scelta teoricamente ottimale per garantire la massima integrità referenziale e consistenza dei dati (proprietà ACID). Tuttavia, la sua adozione avrebbe significato introdurre e manutenere un nuovo sistema nello *stack* aziendale, con un conseguente *overhead* operativo. L'alternativa era allinearsi alla tecnologia già in uso, ovvero un *database* $NoSQL_G$ come **MongoDB**.

Alternative per il Formato di Esportazione Dati

- **JSON/JSONL**: Formato leggibile e flessibile, ottimo per dati gerarchici, ma verboso per dati tabellari e meno diretto da importare in strumenti statistici.
- **Parquet o Arrow**: Formati binari colonnari ad alte *performance*, ideali per *Big Data*, ma illeggibili all'uomo e non immediatamente ispezionabili senza strumenti specifici.

- ***CSV*** $_G$: Formato testuale tabellare universalmente riconosciuto, semplice, leggibile e facilmente manipolabile con librerie standard.

4.4.3 Soluzione adottata e motivazioni

Le scelte sono state guidate da un criterio di pragmatismo e coerenza con il contesto aziendale.

Soluzione per il Database Primario: MongoDB

È stato adottato **MongoDB**. Sebbene altri sistemi potessero offrire vantaggi tecnici specifici, la coerenza con lo *stack* aziendale è stata il criterio predominante. L'infrastruttura tecnologica di Devess si basa già su MongoDB, e questa scelta ha permesso di:

- **Semplificare l'integrazione** e allinearsi alle competenze interne del *team*.
- **Ridurre l'*overhead* operativo**, evitando l'introduzione di un nuovo sistema da gestire.

Soluzione per il Formato di Esportazione: ***CSV*** $_G$

Si è optato per il formato ***CSV*** $_G$ per la sua combinazione unica di semplicità, interoperabilità e, soprattutto, compatibilità con gli *LLM* $_G$. Un fattore decisivo è stata la sua **compatibilità nativa con i modelli linguistici avanzati**: modelli come Claude possono analizzare direttamente il contenuto di un file *CSV* $_G$, permettendo di passare un contesto dati ricco e strutturato all'*agente* in modo efficiente.

4.5 Interfaccia Utente e Visualizzazione Dati

4.5.1 Definizione del problema

Il *front-end*$_G$ ha la sfida di presentare i risultati delle analisi dell'agente *LLM*$_G$ in modo chiaro, intuitivo e interattivo. Poiché le visualizzazioni richieste dal prototipo hanno una struttura predefinita (un *set* fisso di grafici popolati con dati dinamici), il problema era trovare l'architettura più snella ed efficiente per realizzare questo compito, evitando complessità non necessarie.

4.5.2 Analisi delle alternative

Sono state valutate due architetture praticabili per visualizzare dati aggiornati su *dashboard* strutturalmente fisse.

- **Approccio con Piattaforma Esterna (es. Grafana)**: Questa architettura prevede che il *back-end*$_G$ salvi i dati in un *database* intermedio, a cui una piattaforma come Grafana si collega per generare i grafici. Questi verrebbero poi mostrati nel *front-end*$_G$ tramite un componente `<iframe>`. Sebbene robusta, questa soluzione introduce una **complessità infrastrutturale significativa** (un server e un database aggiuntivi da gestire), giudicata un **onere sproporzionato** per il prototipo.
- **Altri Framework JavaScript (Vue.js, Angular)**: Pur essendo alternative valide a React, la loro adozione avrebbe introdotto una frammentazione tecnologica all'interno dello *stack* di Devess, che ha standardizzato l'uso di React, andando contro la strategia aziendale.

4.5.3 Soluzione adottata e motivazioni

È stata scelta la soluzione architetturalmente più diretta: il **rendering nativo con React**. In questo flusso, il *back-end*$_G$ invia i dati dell'analisi direttamente al *front-end*$_G$, che li utilizza per popolare i componenti grafici esistenti, realizzati con librerie come Recharts.

Questa scelta è stata motivata da un'analisi pragmatica dei costi e dei benefici:

- **Minima Complessità Architetturale**: L'approccio evita componenti esterni, rendendo il sistema più semplice da sviluppare, testare e distribuire.
- **Efficienza dei Costi e delle Risorse**: Non richiedendo servizi aggiuntivi, l'architettura minimizza i costi operativi e il carico di manutenzione.
- **Integrazione e Controllo Totale**: Il *rendering* nativo offre un controllo completo sull'aspetto e sull'interattività dei grafici, garantendo una coerenza visiva e un'esperienza utente fluida.

In sintesi, si è scelto l'approccio che offriva il percorso più semplice ed efficiente per raggiungere l'obiettivo, incarnando il principio di evitare complessità non strettamente necessaria.