Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Server MCP per il crawling e l'estrazione strutturata di dati web

Tesi di laurea

Relatore
Prof. Lamberto Ballan

Laureanda
Valeria Baleanu
Matricola 2109911

Anno Accademico 2025-2026

"Nah, I'd win."

— Gojo Satoru

Sommario

Il presente elaborato descrive il lavoro svolto durante il tirocinio curricolare presso l'azienda Riskapp S.r.l.. Il progetto ha avuto come obiettivo principale lo sviluppo di un server *MCP (Model Context Protocol)*, protocollo standard per l'integrazione di strumenti esterni con agenti di intelligenza artificiale.

Il lavoro ha compreso le fasi di analisi dei requisiti, progettazione dell'architettura, codifica e collaudo conclusivo. Sono stati sviluppati strumenti per il recupero del contenuto delle pagine web, la loro conversione in formato *Markdown*, *HTML* e *JSON* strutturato, e un sistema di *caching* intelligente per ottimizzare le prestazioni riducendo le richieste ridondanti.

A supporto del server *MCP* è stata realizzata un'applicazione web completa, composta da un *frontend* (con funzionalità di autenticazione, interfaccia *chatbot*, gestione del profilo e pannello di amministrazione utenti) e un *backend* suddiviso in microservizi.

L'intera applicazione è stata containerizzata tramite *Docker* per garantirne portabilità e scalabilità. Il risultato finale è dunque un server *MCP* funzionante e documentato, distribuibile tramite container e che consente ai modelli di intelligenza artificiale di interrogare fonti web in modo standardizzato e in tempo reale.

"Sono pochi i traguardi che tenacia e perizia non possono raggiungere; le grandi conquiste non si realizzano con la forza, ma con la perseveranza."

— Samuel Johnson

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Lamberto Ballan, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori, mia sorella e il mio fidanzato per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici che, anche se indirettamente, hanno reso questi anni un po' più leggeri e mi hanno permesso di raggiungere questo traguardo con serenità.

Padova, Luglio 2026 Valeria Baleanu

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Il presente capitolo ha l'obiettivo di fornire una descrizione del contesto del progetto di stage e della struttura del presente elaborato. I punti trattati saranno:

- l'azienda ospitante;
- gli obiettivi del progetto e le motivazioni alla base del suo sviluppo;
- la struttura del documento.

1.1 L'azienda

L'azienda scelta per il tirocinio curricolare è RiskApp S.r.l., una realtà italiana specializzata nello sviluppo di soluzioni software e algoritmi per il settore assicurativo, bancario e finanziario. Fondata nel 2015 con sede a Conselve (Padova), l'azienda offre una piattaforma digitale con lo scopo di aiutare le piccole e medie imprese nell'analisi e nella valutazione dei rischi aziendali.

Figura 1.1: Logo di RiskApp S.r.l.

1.2 Il caso di studio

L'efficacia dei modelli di intelligenza artificiale moderni è strettamente legata alla qualità e all'attualità delle informazioni a disposizione. Tuttavia, per loro stessa natura, questi modelli vengono addestrati su un insieme di dati definito in un momento preciso: questo limite strutturale rende la loro conoscenza intrinsecamente statica che dipende dal momento in cui la fase *training* viene eseguita.

Per superare tale limite, è necessario dotare i modelli di un meccanismo che consenta loro di accedere a informazioni aggiornate al momento della richiesta, superando in questo modo il problema temporale imposto dal *training*. È in questo contesto che si inserisce il *Model Context Protocol* (MCP)[g], uno standard che permette ai modelli di intelligenza artificiale di interagire con strumenti e fonti di dati esterni.

Il caso di studio affrontato in questo lavoro di tesi si colloca in questo scenario e nasce dall'esigenza di automatizzare il reperimento di informazioni mirate da siti web. L'obiettivo principale è superare i limiti dei tradizionali sistemi di web crawling[g], che spesso richiedono architetture rigide e difficilmente integrabili con agenti artificiali di ultima generazione.

Su queste basi, l'attività di tirocinio si è focalizzata sull'analisi, progettazione e codifica di una piattaforma in grado di offrire il servizio di web crawling.

1.3 Il software da sviluppare

Il software sviluppato durante il tirocinio è un server MCP, progettato per esporre le funzionalità di web crawling ed estrazione dati da siti web.

Tale server, tuttavia, rappresenta una singola componente di un sistema più ampio: per rendere tale servizio accessibile agli utenti finali, il progetto prevede anche la realizzazione di un'applicazione web composta da un *frontend* e da un *backend*. Il servizio MCP si inserisce all'interno del *backend* ed espone un singolo tool[g], responsabile dell'esecuzione del web crawling su un sito web e della successiva estrazione dei dati in un formato strutturato (*plain text*, HTML[g], Markdown[g] o JSON[g]), restituito poi al chiamante.

1.4 Il ruolo dello *stage* nell'azienda

L'esigenza alla base del progetto è l'automazione del recupero di informazioni dal web, al fine di renderle accessibili in modo standardizzato agli agenti AI utilizzati internamente. In questo modo gli agenti potranno autonomamente chiamare il tool necessario e ricevere il risultato nel formato atteso.

1.5 Organizzazione del testo

L'elaborato si compone di cinque capitoli principali, inclusa l'introduzione.

Il secondo capitolo descrive le attività svolte durante il periodo di tirocinio, con un approfondimento sui problemi affrontati e sulle relative soluzioni implementate.

Il terzo capitolo analizza dettagliatamente i vari requisiti obbligatori, opzionali e desiderabili del progetto.

Il quarto capitolo approfondisce le scelte di progettazione architetturale e la fase di codifica dei componenti sviluppati.

Il quinto capitolo riassume le varie considerazioni finali sull'esperienza di stage evidenziando i possibili sviluppi futuri del sistema.

Riguardo la 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: *parola*[g];
- i termini in lingua straniera di uso non comune o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Descrizione stage

In questo capitolo vengono descritte le attività svolte, l'organizzazione dello stage e l'analisi dei rischi.

2.1 Obiettivi del progetto

Il tirocinio ha come obiettivo principale quello di avvicinare studentesse e studenti al mondo del lavoro, apprendendo le tecnologie e gli strumenti maggiormente richiesti in ambito aziendale. Al termine del tirocinio svolto in RiskApp S.r.l. erano attese le seguenti competenze:

- progettazione e implementazione di un server conforme al MCP per l'esposizione di funzionalità tramite tool dedicati;
- implementazione di tecniche di validazione e prompt engineering per il controllo della qualità dei risultati prodotti dal LLM[g];
- implementazione di un sistema di caching intelligente per la gestione efficiente delle richieste ridondanti;
- sviluppo di un sistema di autenticazione e autorizzazione sicuro basato su JWT[g];
- utilizzo dello strumento di containerizzazione Docker[g] per la gestione e l'orchestrazione dei servizi;
- gestione di database per la persistenza dei dati relativi alle richieste e agli utenti;
- sviluppo di un'interfaccia utente intuitiva e reattiva.

2.2 Vincoli

L'azienda ha definito alcune linee guida per garantire la portabilità del prodotto finale e l'interoperabilità con i software aziendali interni. I vincoli individuati sono i seguenti:

- il prodotto deve essere sviluppato usando il framework[g] React per il *frontend*;
- il prodotto deve essere sviluppato usando il framework FastAPI per il *backend*;
- il sistema deve essere containerizzato tramite Docker;
- le componenti del sistema devono comunicare tramite API[g].

2.3 Pianificazione

All'inizio del tirocinio è stato redatto il piano di lavoro per la ripartizione approssimativa delle attività su base settimanale. La pianificazione prevista è dunque la seguente:

- **Prima settimana**
  - incontro con le persone coinvolte nel progetto per discutere i vari requisiti;
  - studio e formazione sulle tecnologie adottate, quali FastMCP e FastAPI.
- **Seconda settimana**
  - studio e formazione sulle tecnologie adottate;
  - analisi dei requisiti.
- **Terza settimana**
  - sviluppo del prototipo;
  - inizio progettazione;
  - inizio pianificazione.
- **Quarta settimana**
  - configurazione dell'ambiente di sviluppo con Docker;
  - implementazione del componente server MCP.
- **Quinta settimana**
  - implementazione della conversione delle pagine in formato Markdown e HTML;
  - implementazione del sistema di caching intelligente.
- **Sesta settimana**
  - gestione degli errori e casi limite;
  - ottimizzazione delle performance del sistema.
- **Settima settimana**
  - sviluppo dei *test*;
  - collaudo e verifica;
- **Ottava settimana**
  - completamento del collaudo e correzione *bug*;
  - stesura della documentazione tecnica finale.

2.3.1 Prima settimana

Durante la prima settimana di lavoro ha avuto luogo l'incontro iniziale con i *tutor* aziendali per un'analisi approfondita del progetto e la definizione dei requisiti principali. Tale incontro è stato fondamentale per definire con precisione gli obiettivi e delineare i vincoli tecnologici e operativi. Parallelamente, è iniziata una fase di studio individuale focalizzata sulla lettura della documentazione relativa ai framework *FastMCP* e *FastAPI*. Per velocizzare l'apprendimento di tali tecnologie ho sviluppato dei piccoli progetti guidati in *Python* disponibili nella documentazione ufficiale (*QuickStart*), Tali progetti erano volti alla comprensione pratica delle tecnologie studiate. In particolare:

- riguardo **FastAPI** ho imparato a implementare:
  - funzioni `async`/`await`;
  - classi con `@dataclass`;
  - *streaming* di dati in formato JSON per il SSE[g];
  - WebSocket;
  - i principali metodi HTTP (`GET`, `POST`, `PUT`, `DELETE`);
- riguardo **FastMCP**, ho imparato a implementare:
  - tool MCP;
  - *resources* MCP;
  - *prompts* MCP;

2.3.2 Seconda settimana

La seconda settimana è stata dedicata al consolidamento delle competenze tecnologiche acquisite nei giorni precedenti. Per una comprensione più solida del progetto, si è proceduto con la redazione del documento di Analisi dei Requisiti. Tale attività ha permesso di formalizzare tutti i requisiti, che sono stati suddivisi in requisiti funzionali, di qualità e di vincolo e a sua volta scomposti in obbligatori, opzionali e desiderabili. Durante questo periodo sono stati svolti alcuni incontri con i *tutor* aziendali per chiarire alcuni dubbi e aggiungere nuovi requisiti, i quali hanno permesso di espandere il progetto.

2.3.3 Terza settimana

Nel corso della terza settimana l'attenzione si è spostata verso le attività di sviluppo di un *Proof of Concept*, volto a permettere all'azienda di valutare la mia comprensione degli obiettivi e dell'attuabilità del progetto.
Il prototipo realizzato ha previsto lo sviluppo del server MCP che esponeva un tool di web crawling. Tale tool richiedeva come parametri il nome del sito web da cui eseguire il web crawling, la domanda dell'utente in linguaggio naturale e il tipo di formato desiderato per la generazione della risposta.

Il sistema avviava un processo di web crawling che, ricorsivamente, estraeva l'HTML della pagina, lo riassumeva e tentava di rispondere alla domanda dell'utente. Se le informazioni ricavate fino a quel punto non erano sufficienti a rispondere al quesito, il sistema prelevava tutti i *link* ai siti del medesimo dominio web, selezionava i più pertinenti (al più tre) e proseguiva la ricerca. Per evitare una ricorsione infinita o un numero eccessivo di chiamate ricorsive il sistema definiva due parametri: l'ampiezza massima (il numero massimo di *link* selezionabili ad ogni livello di ricorsione) e profondità massima (numero massimo di livelli di ricorsione). Successivamente è stata sviluppata anche un'interfaccia utente per l'inserimento dei parametri e la visualizzazione della risposta generata. Infine, è stato implementato un processo di *streaming* tramite SSE, per mostrare nella pagina web i singoli passaggi svolti durante la ricorsione (estrazione della pagina, riassunto del contenuto, estrazione di tutti i *link*, scelta dei *link* più pertinenti).

Il prototipo sviluppato ha permesso di dimostrare concretamente la fattibilità dell'approccio proposto, fornendo all'azienda una prima base tangibile su cui valutare la direzione del progetto e indirizzare le fasi di sviluppo successive.

In base ai riscontri del *Proof of Concept*, è iniziata la pianificazione di dettaglio dell'architettura del sistema, prima per il *backend* e successivamente per il *frontend*. In particolare, sono state effettuate le seguenti scelte architetturali:

- suddivisione del *backend* in microservizi indipendenti, ciascuno responsabile di un dominio applicativo specifico (autenticazione, gestione utenti, storico delle richieste, ricerca e server MCP) con l'obiettivo di favorire la scalabilità e la manutenibilità del sistema;
- adozione dell'architettura esagonale (*hexagonal architecture*) all'interno di ciascun microservizio al fine di disaccoppiare la logica di business dai dettagli implementativi (come database, LLM, API esterne) e favorire la testabilità del codice;
- adozione del pattern MVVM[g]per il *frontend* allo scopo di separare la logica di presentazione dalla gestione dello stato dell'applicazione.

2.3.4 Quarta settimana

La quarta settimana ha segnato l'inizio della fase di codifica dell'infrastruttura, partendo dal componente *backend*. Il primo obiettivo è stato la configurazione dell'ambiente di sviluppo attraverso la containerizzazione con Docker e la suddivisione dei container per microservizi, al fine di garantire fin dall'inizio la portabilità e scalabilità del software, uno dei vincoli principali imposti dall'azienda. In seguito è stato definito la base di dati dell'applicazione, al fine di garantire la persistenza dei dati relativi agli utenti (profilo, permessi, chat) e alle richieste effettuate al LLM.
Infine, è stata avviata la codifica dei vari microservizi del *backend*. Il processo di codifica di tali microservizi è stato protratto per le successive settimane e sono stati sviluppati nel seguente ordine:

1. **server MCP service**: rappresenta il vero e proprio servizio MCP che espone il tool di *crawl* e per ogni passaggio effettuato, invia al *frontend* un *event* tramite SSE;
2. **search service**: rappresenta il servizio dedicato alla gestione dei messaggi dell'utente il linguaggio naturale. Il microservizio, dato il contesto, comprende quale tool è richiesto dall'utente, estrae i parametri necessari e chiama l'*endpoint* corretto del microservizio Server MCP;
3. **auth service**: rappresenta il servizio dedicato all'autenticazione dell'utente e alla gestione dei JWT (validazione e creazione di *access token* e *refresh token*, salvataggio e revoca del *refresh token*);
4. **user service**: rappresenta il servizio dedicato alla creazione e alla gestione degli utenti;
5. **history service**: rappresenta il servizio dedicato alla gestione e salvataggio dei messaggi delle *chat* tra l'utente e l'LLM.

Dalla quarta settimana la pianificazione prevista dal piano di lavoro non è stata più seguita: dopo l'analisi dei requisiti sono sorte nuove funzionalità importanti e complesse che non erano state dichiarate all'inizio. Il prodotto finale avrebbe dunque richiesto più lavoro su vari fronti, diversi dalla sola gestione di un servizio MCP. Durante tale periodo è stato dunque fondamentale una ripianificazione completa delle settimane a seguire in maniera tale da adattarla alle nuove specifiche.

2.3.5 Quinta settimana

Durante la quinta settimana si è continuato il processo di codifica *backend* avviato nella quarta settimana. In particolare:

- è stato completato il microservizio relativo al server MCP;

- è stato avviato e completato *search service*;
- è stato avviato *auth service*.

2.3.6 Sesta settimana

Nella sesta settimana si è continuato il processo di codifica, in particolare:

- è stato completato il microservizio *auth service*;
- sono stato avviati e completati *user service* e *history service*.

Avendo concluso lo sviluppo del *backend*, ho iniziato la codifica relativa alla parte del *frontend*, la quale è stata suddivisa in base alle funzionalità fornite. Sono state dunque implementate le seguenti pagine web:

- **login** per effettuare l'accesso al sistema tramite le credenziali *username* e *password*;
- **chatbot** per inviare messaggi al sistema e visualizzare la risposta generata con i vari passaggi del processo di web crawling;
- **gestione utenti**, dedicato ai soli utenti *admin*, permette di creare e gestire nuovi utenti all'interno del sistema;
- **profilo utente** per visualizzare e modificare le informazioni dell'utente;
- **storico delle *chat*** per visualizzare le varie conversazioni passate.

2.3.7 Settima settimana

La settima settimana è stata dedicata al *refactoring*: partendo dal *backend* fino a raggiungere il *frontend*, sono state analizzate tutte le componenti e le funzioni implementate e sono state ottimizzate, al fine di renderle più efficienti dal punto di vista di complessità, velocità e utilizzo di risorse. Un'attività di particolare rilievo ha riguardato l'implementazione del sistema di *tracing* all'interno del servizio MCP, realizzato mediante la piattaforma *LangSmith*. Tale funzionalità consente di monitorare e analizzare in modo dettagliato le chiamate effettuate ai modelli di intelligenza artificiale impiegati dal sistema.

Sono state poi implementate piccole *feature* aggiuntive per migliorare l'*user experience*, come l'utilizzo di icone nei bottoni, il tema chiaro/scuro e la possibilità di copiare o scaricare la risposta generata. Infine sono stati sviluppati i vari *test* relativi a tutte le componenti del sistema per validare il loro corretto funzionamento e valutare la conformità rispetto ai requisiti definiti in fase di analisi.

2.3.8 Ottava settimana

Durante l'ultima settimana sono stati corretti alcuni *bug* relativi alla visualizzazione dei messaggi nelle *chat* e si è concluso ufficialmente il progetto con il collaudo finale da parte dei *tutor* aziendali.

2.3.9 Pianificazione in retrospettiva

L'esperienza di tirocinio si è rivelata particolarmente formativa soprattutto dal punto di vista tecnico. Sebbene la pianificazione iniziale abbia costituito un utile punto di partenza per organizzare le attività, il progetto ha subito un'evoluzione importante durante il suo sviluppo. Dopo la fase di analisi dei requisiti, avviata nella seconda settimana, sono infatti emerse nuove esigenze e funzionalità che hanno portato a una revisione significativa degli obiettivi iniziali e della pianificazione prevista.

Questa situazione ha evidenziato come, all'interno di un contesto aziendale reale, i requisiti possano evolvere nel tempo in funzione delle necessità del cliente, delle opportunità individuate durante lo sviluppo e dei risultati ottenuti nelle fasi preliminari del progetto. Di conseguenza, è stato necessario adattare costantemente il piano di lavoro, ridefinendo priorità e attività da svolgere. Tale esperienza ha permesso di comprendere l'importanza della flessibilità e della capacità di gestione del cambiamento, competenze fondamentali nello sviluppo software moderno.

Dal punto di vista tecnico, il progetto ha rappresentato un'importante occasione di approfondimento di tecnologie e paradigmi architetturali non affrontati in modo approfondito durante il percorso universitario. In particolare, ho acquisito competenze nell'utilizzo di Python e dei relativi framework FastAPI e FastMCP, nella progettazione di architetture a microservizi, nell'applicazione dell'architettura esagonale e nell'utilizzo del pattern MVVM per lo sviluppo *frontend*. Ho inoltre maturato esperienza nella containerizzazione delle applicazioni mediante Docker e nella progettazione di sistemi che integrano modelli linguistici di grandi dimensioni (LLM).

Un altro aspetto particolarmente significativo è stato il confronto continuo con i *tutor* aziendali. Le revisioni periodiche e gli incontri mirati sulle scelte progettuali hanno contribuito a migliorare la qualità delle soluzioni sviluppate e mi hanno permesso di imparare molte tecnologie e tecniche nuove. Questo confronto costante ha inoltre favorito lo sviluppo delle capacità comunicative.

Il raggiungimento degli obiettivi finali e la realizzazione di un prodotto completo, composto da un'infrastruttura *backend* a microservizi e da un'interfaccia *frontend* dedicata, hanno rappresentato una conferma concreta delle competenze acquisite durante il percorso universitario e di quelle sviluppate nel corso del tirocinio. L'esperienza ha quindi contribuito in maniera significativa alla mia crescita professionale, fornendomi una visione più completa delle attività necessarie per progettare, sviluppare, testare e distribuire un'applicazione software all'interno di un contesto aziendale reale.

2.4 Organizzazione del lavoro

2.4.1 Metodologia adottata

Per la gestione e l'esecuzione delle attività durante tutto il periodo di stage è stata scelta una metodologia di sviluppo di tipo *iterativo-incrementale* basata su *timebox* settimanali. Questo approccio ha permesso di tradurre la pianificazione strategica di alto livello in obiettivi operativi intermedi e costantemente verificabili. La misura dell'avanzamento del progetto è stata determinata dal progressivo rilascio di incrementi software stabili e funzionanti, garantendo al contempo la flessibilità necessaria per ripianificare gli obiettivi in base alle esigenze emerse in corso d'opera.

Il modello iterativo-incrementale si fonda sul principio del miglioramento continuo del prodotto attraverso cicli ripetuti di analisi, progettazione, implementazione e verifica. Differisce dai modelli sequenziali poiché l'applicazione non viene sviluppata in un'unica soluzione finale, ma cresce per "incrementi" successivi. Ogni iterazione produce una versione del software che aggiunge nuove funzionalità o ottimizza quelle esistenti, consentendo di integrare tempestivamente i *feedback* degli *stakeholder* e mitigare i rischi tecnici.

Il ciclo di vita del progetto di tirocinio è stato mappato sulle quattro macro-fasi logiche caratteristiche di questo modello di sviluppo:

- **Inception**: svoltasi prevalentemente durante la prima settimana, ha compreso la definizione del perimetro del progetto, l'analisi preliminare dei vincoli tecnologici (*Docker*, *Python* con *FastAPI* e *FastMCP*, *React*, *PostgreSQL*) e l'allineamento iniziale con i *tutor* aziendali per delineare gli obiettivi ad alto livello;

- **Elaboration**: coincidente con la seconda e la terza settimana, in cui si è proceduto alla formalizzazione del documento di Analisi dei Requisiti e alla successiva progettazione architetturale, validata attraverso la realizzazione di un *Proof of Concept*;
- **Construction**: estesa dalla quarta alla settima settimana, ha rappresentato la fase a più alta intensità di sviluppo. In questo periodo l'architettura a microservizi del *backend* e l'interfaccia *frontend* in *React* sono state progressivamente codificate, integrate e containerizzate tramite *Docker*. Sono stati sviluppati i *test* automatici ed è stata svolta l'attività di *refactoring* di tutte le componenti di sistema;
- **Transition**: focalizzata nell'ultima settimana, ha riguardato l'ottimizzazione delle performance e la consegna finale del prodotto ai *tutor* aziendali.

2.4.2 Analisi comparativa e motivazioni della scelta

Gestione del cambiamento

La scelta di questa metodologia è stata determinata dalla natura innovativa e dinamica del progetto, incentrato sull'utilizzo del protocollo MCP e sull'integrazione di modelli linguistici di grandi dimensioni (LLM).

L'efficacia della scelta iterativa si è manifestata concretamente a partire dalla quarta settimana: l'evoluzione dei requisiti ha reso obsoleta la pianificazione iniziale. In un framework rigido, tale variazione avrebbe comportato una seria minaccia per la stabilità e le tempistiche del progetto; l'approccio iterativo ha invece permesso di avviare una ripianificazione delle settimane rimanenti, rimodulando i confini dei *timebox* e garantendo lo sviluppo delle nuove funzionalità senza compromettere la qualità del software o i tempi di consegna.

Confronto con altre metodologie

Per valutare la qualità nel contesto del progetto, la metodologia adottata è stata messa a confronto con i principali paradigmi alternativi:

- **Modello Waterfall (A Cascata)**: questo approccio prevede una sequenza rigida di fasi in cui i requisiti devono essere interamente definiti e congelati all'inizio. Applicare il modello a cascata in questo scenario sarebbe stato fallimentare, poiché l'evoluzione dei requisiti avrebbe richiesto un ritorno alle fasi iniziali, provocando ritardi insostenibili.
- **Scrum puro**: sebbene Scrum sia un'ottima metodologia agile per gestire requisiti fluttuanti, la sua struttura cerimoniale rigorosa (comprendente *Daily Standup*, *Sprint Planning*, *Review* e *Retrospective*) unita alla definizione formale di ruoli rigidi (*Scrum Master*, *Product Owner*) avrebbe generato un sovraccarico organizzativo ingiustificato per un progetto svolto individualmente.
- **Kanban**: pur offrendo flessibilità e concentrandosi sul flusso continuo di lavoro, Kanban non impone scadenze temporali rigide o cicli di pianificazione predefiniti. L'assenza di *timebox* strutturati avrebbe reso complessa la verifica periodica degli obiettivi soddisfatti.

2.4.3 Metodologia operativa e flussi di comunicazione

Le attività settimanali sono state governate da due dinamiche principali di sincronizzazione con la struttura aziendale:

1. **Supporto quotidiano e comunicazione asincrona**: anche durante le giornate svolte da remoto, ero sempre in contatto con i *tutor* aziendali tramite l'applicazione

di messaggistica *Slack*. Tale canale ha permesso di mantenere un flusso informativo costante con i *tutor*, facilitando la rapida risoluzione di problemi o di domande relative al progetto;

2. **Allineamento finale settimanale**: un incontro della durata di circa 20 minuti programmato al termine di ogni ciclo. Durante la sessione veniva effettuato un controllo dei risultati ottenuti durante la settimana e veniva pianificato l'obiettivo del ciclo successivo.

2.4.4 Esempio di iterazione settimanale

A titolo esemplificativo, nella Tabella 2.1 viene riportata la scheda di gestione utilizzata per documentare e tracciare le attività della terza settimana, periodo in cui è stato sviluppato il prototipo del sistema di web crawling.

Elemento	**Descrizione**
Settimana	Settimana 3
Goal	Sviluppo di un *Proof of Concept* (PoC) funzionante, finalizzato alla verifica della fattibilità del server MCP.
Attività pianificate in ordine di priorità	1. Implementazione base del server *MCP* con esposizione di un *tool* di *crawling*; 2. Sviluppo della logica di estrazione, riassunto e selezione *link* delle pagine HTML; 3. Formattazione della risposta generata nei formati HTML, JSON, plain text e Markdown con vincoli di ampiezza e profondità massima; 4. Realizzazione di un'interfaccia utente di base con inserimento URL dei sito web, inserimento domanda e visualizzazione della risposta finale; 5. Implementazione dello *streaming* degli stati tramite *Server-Sent Events* (SSE); 6. Visualizzazione via *frontend* dei passaggi in tempo reale.
Esito	Obiettivo raggiunto.
Feedback	Valutazione positiva. Viene consigliato di rendere *built-in* (statici) i parametri di massima ampiezza e profondità.

Tabella 2.1: Documentazione di un'iterazione settimanale (Settimana 3).

2.5 Analisi preventiva dei rischi

Durante la fase di analisi iniziale sono stati individuati alcuni possibili rischi a cui si potrà andare incontro. Si è quindi proceduto a elaborare delle possibili soluzioni per far fronte a tali rischi.

1. Elevata curva di apprendimento tecnologico
Descrizione: L'adozione di un ampio *stack* tecnologico comprendente framework e paradigmi non approfonditi nel percorso accademico (FastAPI, protocollo MCP, architettura esagonale e Docker) rischia di sottrarre tempo prezioso alla fase più dispendiosa, ovvero la codifica.

Soluzione: Pianificazione di una fase iniziale di formazione teorico-pratica con sviluppo di progetti guidati (*QuickStart*) e di un *Proof of Concept* per validare le competenze prima dello sviluppo definitivo.

2. Volatilità e instabilità dei requisiti
Descrizione: Trattandosi di un'applicazione legata a tecnologie emergenti in un contesto aziendale reale, i requisiti iniziali potrebbero subire variazioni o ampliamenti in corso d'opera, invalidando la pianificazione strategica originaria.
Soluzione: Adozione di una metodologia iterativo-incrementale basata su *timebox* settimanali e canali di comunicazione costanti (*Slack* e allineamenti), permettendo una tempestiva ripianificazione degli obiettivi.

3. Latenza e limitazioni nell'integrazione di LLM e crawling
Descrizione: L'interazione con i modelli linguistici e i processi ricorsivi di *web crawling* potrebbero introdurre colli di bottiglia e problemi di *rate limiting* delle *API*.
Soluzione: Definizione di parametri rigidi di ampiezza e profondità massima per la ricorsione, implementazione di un sistema di *caching* intelligente e utilizzo della piattaforma *LangSmith* per il tracciamento e il monitoraggio continuo delle chiamate.

4. Non-determinismo delle risposte del modello linguistico (LLM)
Descrizione: I modelli linguistici di grandi dimensioni possono generare risposte imprevedibili o strutturalmente non omogenee (comunemente chiamate "allucinazioni") a parità di *input*, rischiando di compromettere la consistenza dei dati inviati al *frontend* o la corretta esecuzione dei tool basati sul protocollo MCP.
Soluzione: Implementazione di tecniche stringenti di *prompt engineering*, definizione di vincoli sintattici rigorosi sull'*output* del modello (richiedendo formati strutturati, come JSON) e aggiunta di validazione dei dati per intercettare e gestire tempestivamente eventuali risposte non conformi.

5. Complessità di integrazione dell'architettura a microservizi
Descrizione: La scomposizione del *backend* in microservizi indipendenti introduce complessità nella comunicazione tra componenti e nella gestione della persistenza.
Soluzione: Configurazione e orchestrazione dell'ambiente di sviluppo tramite *Docker* fin dalle prime fasi e sviluppo sequenziale e incrementale dei singoli microservizi.

2.6 Requisiti e obiettivi

In questa sezione vengono formalizzati gli obiettivi prefissati all'inizio dell'esperienza di tirocinio, classificati in base alla loro priorità di realizzazione.

2.6.1 Obiettivi obbligatori

Gli obiettivi obbligatori rappresentano i requisiti minimi e indispensabili per il successo del progetto. Essi comprendono le componenti architetturali e le funzionalità primarie di interazione con i modelli linguistici:

- **Progettazione e implementazione del server MCP**: Sviluppo di un server conforme allo standard *Model Context Protocol* (MCP) mediante l'utilizzo del framework *FastMCP*, finalizzato all'esposizione di funzionalità e strumenti dedicati all'ecosistema di intelligenza artificiale.
- **Sviluppo del modulo di crawling parametrico**: Realizzazione di un motore di *web crawling* ricorsivo in grado di estrarre e riassumere il contenuto informativo di pagine web in linguaggio naturale, governato da vincoli rigidi di ampiezza e profondità massima per prevenire cicli infiniti o sovraccarichi computazionali.

- **Ingegnerizzazione dei prompt e validazione dell'output AI**: Applicazione di tecniche avanzate di *prompt engineering* e introduzione di uno strato software di validazione sintattica e semantica, volto a garantire la qualità e la coerenza dei risultati prodotti dal *Large Language Model* (LLM).
- **Architettura del backend a microservizi**: Progettazione e codifica dei servizi di *backend* (gestione utenti, autenticazione, storico chat, servizi di ricerca e server MCP) tramite il framework *FastAPI*, strutturati secondo i dettami dell'architettura esagonale per disaccoppiare la logica di business dalle dipendenze tecnologiche.
- **Sviluppo di un'interfaccia utente reattiva**: Realizzazione del *frontend* dell'applicazione mediante il framework *React* (adottando il pattern MVVM), che includa una pagina per l'interazione in stile chatbot con il modello e la visualizzazione in tempo reale, tramite SSE, dei singoli passaggi logici eseguiti dal *crawler*.
- **Gestione della sicurezza e dell'autenticazione**: Implementazione di un sistema sicuro di controllo degli accessi basato su JWT, che regoli l'emissione, la validazione e la revoca di *access token* e *refresh token*, associando permessi specifici in base al ruolo dell'utente (es. utente standard ed amministratore).
- **Containerizzazione e orchestrazione dell'infrastruttura**: Utilizzo della piattaforma *Docker* per l'isolamento di ciascun microservizio e del database relazionale, garantendo la parità degli ambienti, la portabilità del software e il rispetto dei vincoli di interoperabilità aziendali.
- **Verifica e documentazione**: Conduzione di *test* di unità e di integrazione per validare il corretto funzionamento del *routing* dei *MCP tools* e stesura della documentazione tecnica finale.

2.6.2 Obiettivi desiderabili

Gli obiettivi desiderabili comprendono tutte quelle funzionalità aggiuntive, attività di ottimizzazione e strumenti di monitoraggio volti ad elevare la qualità del prodotto finale e la sua esperienza d'uso:

- **Implementazione di un sistema di caching intelligente**: Sviluppo di un meccanismo di persistenza temporanea delle richieste per intercettare e gestire in modo efficiente i quesiti ridondanti formulati dagli utenti, abbattendo i tempi di latenza complessivi del sistema e riducendo il consumo di *token* verso le API del modello linguistico.
- **Integrazione di sistemi di tracing avanzati**: Introduzione e configurazione della piattaforma *LangSmith* all'interno del servizio MCP, allo scopo di monitorare, tracciare ed analizzare nel dettaglio il flusso delle chiamate e le performance dei modelli di intelligenza artificiale impiegati.
- **Funzionalità avanzate di gestione e persistenza**: Estensione delle capacità del sistema attraverso la creazione di un pannello amministrativo per la gestione delle utenze, la personalizzazione del profilo utente e la memorizzazione persistente su database dello storico delle conversazioni.
- **Esportazione locale dei dati generati**: Sviluppo di una funzionalità che consenta all'utente di salvare e scaricare la risposta generata dall'assistente virtuale in un file locale (in conformità con i requisiti di utilità aziendale), gestendo in modo robusto gli eventuali errori di scrittura del file.

- **Ottimizzazione dell'User Experience (UX)**: Introduzione di accorgimenti grafici e funzionali volti a migliorare l'usabilità dell'interfaccia web, tra cui il supporto del tema chiaro/scuro, elementi visivi iconografici immediati e opzioni di copia rapida negli appunti o di *download* per i testi generati.

Capitolo 3

Analisi dei requisiti

In questo capitolo vengono descritti in dettaglio tutti i requisiti funzionali, di qualità e di vincolo.

3.1 Casi d'uso

Un caso d'uso è la descrizione dettagliata, tramite un diagramma UML e una descrizione testuale, di un insieme di scenari che hanno uno scopo comune per un attore all'interno del sistema.

3.1.1 Attori principali

L'attore principale è l'**utente**, suddiviso in utente base e admin.

Utente	Utente generico registrato nel sistema
Admin	Amministratore di sistema
Utente base	Utente con permessi standard

Tabella 3.1: Attori principali

3.1.2 Attori secondari

L'attore secondario è il modello di intelligenza artificiale (**LLM**), il quale viene utilizzato dal sistema per svolgere le funzionalità definite nei tool esposti, tra cui l'azione di web crawling dei vari siti web.

LLM	Modello di intelligenza artificiale per l'elaborazione

Tabella 3.2: Attori secondari

3.2 Lista dei casi d'uso

3.2.1 UC1 - Autenticazione

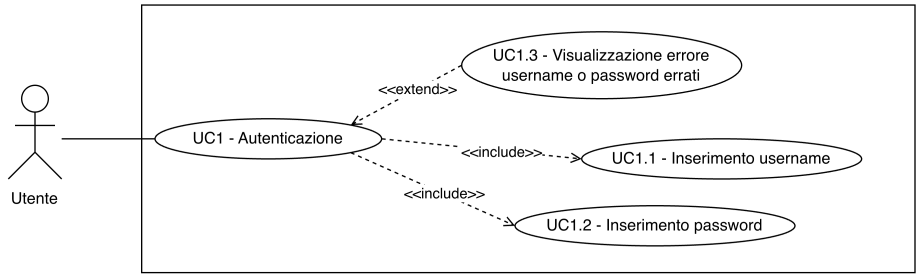

Figura 3.1: Diagramma UC1

Attori	Utente
Pre-condizioni	• Il sistema è attivo; • l'utente si trova nella pagina di autenticazione.
Post-condizioni	L'utente è autenticato nel sistema.

Scenario principale	1. L'utente si autentica con le proprie credenziali:
	  • inserimento *username* (§3.2.2);
	  • inserimento password (§3.2.3).
	2. l'utente conferma l'operazione.

Inclusioni	• UC1.1 §3.2.2 • UC1.2 §3.2.3
Scenari alternativi	L'utente non è registrato nel sistema o la password è errata (§3.2.4).
Estensioni	UC1.3 §3.2.4

3.2.2 UC1.1 - Inserimento username

Attori	Utente
Pre-condizioni	L'utente sta eseguendo l'autenticazione.
Post-condizioni	Il sistema riceve l'*username* dell'utente.
Scenario principale	L'utente inserisce l'*username*.

3.2.3 UC1.2 - Inserimento password

Attori	Utente
Pre-condizioni	L'utente sta eseguendo l'autenticazione.

Post-condizioni	Il sistema riceve la password dell'utente.
Scenario principale	L'utente inserisce la password.

3.2.4 UC1.3 - Visualizzazione errore username o password errati

Attori	Utente
Pre-condizioni	L'utente ha inserito uno *username* che non corrisponde ad un utente registrato nel sistema oppure la password è errata per l'*username* inserito.
Post-condizioni	Il sistema non autentica l'utente.
Scenario principale	Il sistema mostra un messaggio di errore.

3.2.5 UC2 - Logout

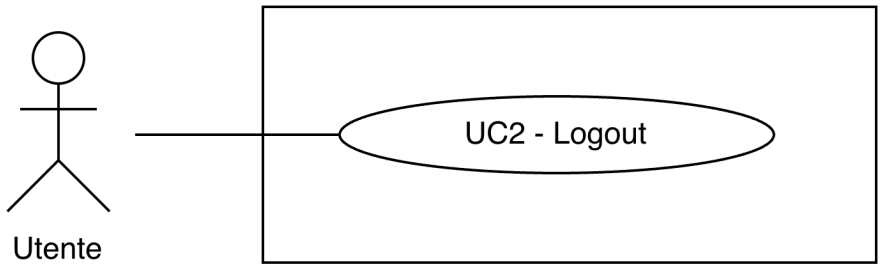

Figura 3.2: Diagramma UC2

Attori	Utente
Pre-condizioni	• Il sistema è attivo; • l'utente è autenticato nel sistema; • l'utente si trova nella pagina di gestione del profilo.
Post-condizioni	Il sistema effettua il *logout* dell'utente, riportandolo alla pagina di autenticazione.
Scenario principale	L'utente seleziona l'opzione relativa al *logout*.

3.2.6 UC3 - Creazione utente

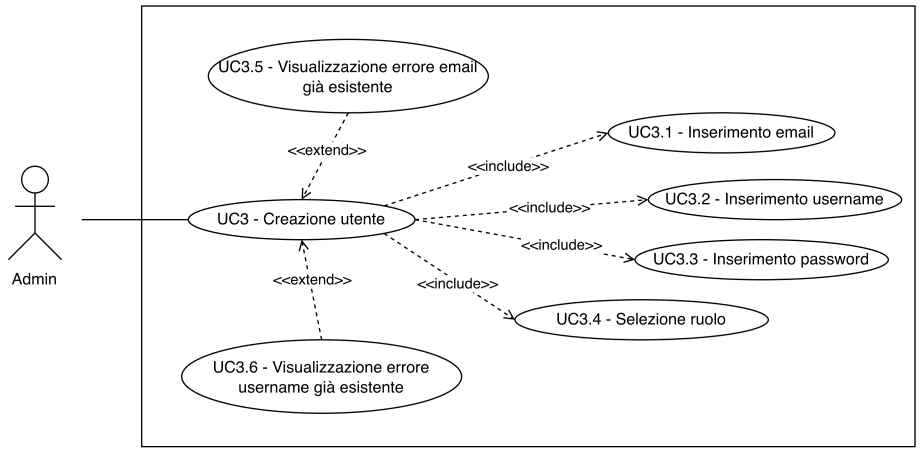

Figura 3.3: Diagramma UC3

Attori	Admin
Pre-condizioni	• Il sistema è attivo; • l'admin è autenticato nel sistema; • l'admin si trova nella pagina di gestione degli utenti.
Post-condizioni	L'admin crea un nuovo utente.

Scenario principale	1. L'admin crea un nuovo utente tramite:
	  • inserimento email (§3.2.7);
	  • inserimento *username* (§3.2.8);
	  • inserimento password (§3.2.9);
	  • selezione ruolo dell'utente (§3.2.10).
	2. l'admin conferma l'operazione.

Inclusioni	• UC3.1 §3.2.7 • UC3.2 §3.2.8 • UC3.3 §3.2.9 • UC3.4 §3.2.10
Scenari alternativi	• L'email è già esistente nel sistema (§3.2.11); • l'*username* è già esistente nel sistema (§3.2.12).
Estensioni	• UC3.5 §3.2.11 • UC3.6 §3.2.12

3.2.7 UC3.1 - Inserimento email

Attori	Admin
Pre-condizioni	L'admin sta eseguendo la creazione di un nuovo utente.
Post-condizioni	Il sistema riceve l'email del nuovo utente.
Scenario principale	L'admin inserisce l'email.

3.2.8 UC3.2 - Inserimento username

Attori	Admin
Pre-condizioni	L'admin sta eseguendo la creazione di un nuovo utente.
Post-condizioni	Il sistema riceve l'*username* del nuovo utente.
Scenario principale	L'admin inserisce l'*username*.

3.2.9 UC3.3 - Inserimento password

Attori	Admin
Pre-condizioni	L'admin sta eseguendo la creazione di un nuovo utente.
Post-condizioni	Il sistema riceve la password del nuovo utente.
Scenario principale	L'admin inserisce la password.

3.2.10 UC3.4 - Selezione ruolo

Attori	Admin
Pre-condizioni	L'admin sta eseguendo la creazione di un nuovo utente.
Post-condizioni	Il sistema riceve la scelta del ruolo dell'utente.
Scenario principale	L'admin seleziona il ruolo da assegnare tra "utente base" e "admin".

3.2.11 UC3.5 - Visualizzazione errore email già esistente

Attori	Admin
Pre-condizioni	L'admin ha inserito una email che è già utilizzata all'interno del sistema.
Post-condizioni	Il sistema non registra l'utente.
Scenario principale	Il sistema mostra un messaggio di errore.

3.2.12 UC3.6 - Visualizzazione errore *username* già esistente

Attori	Admin

Pre-condizioni	L'admin ha inserito uno *username* che è già utilizzato all'interno del sistema.
Post-condizioni	Il sistema non registra l'utente.
Scenario principale	Il sistema mostra un messaggio di errore.

3.2.13 UC4 - Eliminazione utente

Attori	Admin
Pre-condizioni	• Il sistema è attivo; • l'admin è autenticato nel sistema; • l'admin si trova nella pagina di gestione degli utenti.
Post-condizioni	Il sistema elimina il profilo dell'utente selezionato.
Scenario principale	1. l'admin seleziona l'utente che vuole eliminare; 2. l'admin seleziona l'opzione di eliminazione dell'utente; 3. l'admin visualizza una finestra di dialogo per confermare la scelta di eliminazione del profilo, quindi conferma.

3.2.14 UC5 - Visualizzazione profilo

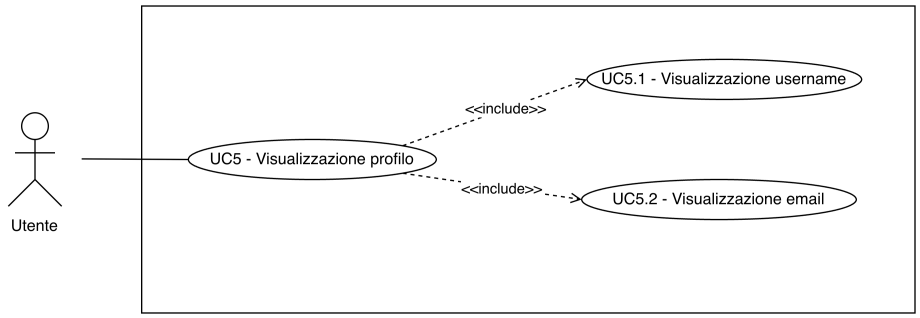

Figura 3.4: Diagramma UC5

Attori	Utente
Pre-condizioni	• Il sistema è attivo; • l'utente è autenticato nel sistema; • l'utente si trova nella pagina di gestione del profilo.
Post-condizioni	L'utente visualizza il proprio profilo.
Scenario principale	L'utente visualizza le informazioni relative al profilo: • visualizzazione *username* (§3.2.15); • visualizzazione email (§3.2.16).

Inclusioni	• UC5.1 §3.2.15 • UC5.2 §3.2.16

3.2.15 UC5.1 - Visualizzazione username

Attori	Utente
Pre-condizioni	L'utente sta visualizzando il proprio profilo.
Post-condizioni	L'utente visualizza l'*username*.
Scenario principale	L'utente visualizza l'*username*.

3.2.16 UC5.2 - Visualizzazione email

Attori	Utente
Pre-condizioni	L'utente sta visualizzando il proprio profilo.
Post-condizioni	L'utente visualizza l'email.
Scenario principale	L'utente visualizza l'email.

3.2.17 UC6 - Modifica profilo

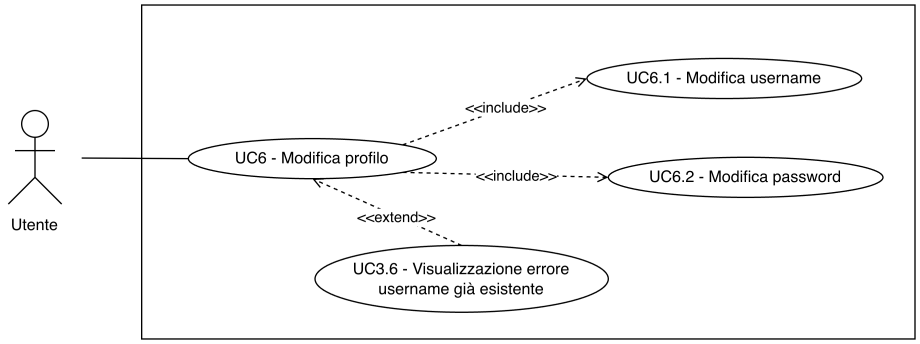

Figura 3.5: Diagramma UC6

Attori	Utente
Pre-condizioni	• Il sistema è attivo; • l'utente è autenticato nel sistema; • l'utente si trova nella pagina di gestione del profilo.
Post-condizioni	L'utente modifica le proprie credenziali.

Scenario principale	1. L'utente seleziona l'opzione di modifica del profilo; 2. l'utente modifica le credenziali: • modifica *username* (§3.2.18); • modifica password (§3.2.19). 3. l'utente conferma l'operazione.
Inclusioni	• UC6.1 §3.2.18 • UC6.2 §3.2.19
Scenari alternativi	• l'*username* è già esistente nel sistema (§3.2.12).
Estensioni	• UC3.6 §3.2.12

3.2.18 UC6.1 - Modifica username

Attori	Utente
Pre-condizioni	L'utente sta eseguendo la modifica del profilo.
Post-condizioni	Il sistema riceve l'*username* dell'utente.
Scenario principale	L'utente inserisce l'*username*.

3.2.19 UC6.2 - Modifica password

Attori	Utente
Pre-condizioni	L'utente sta eseguendo la modifica del profilo.
Post-condizioni	Il sistema riceve la password dell'utente.
Scenario principale	L'utente inserisce la password.

3.2.20 UC7 - Invio messaggio in linguaggio naturale

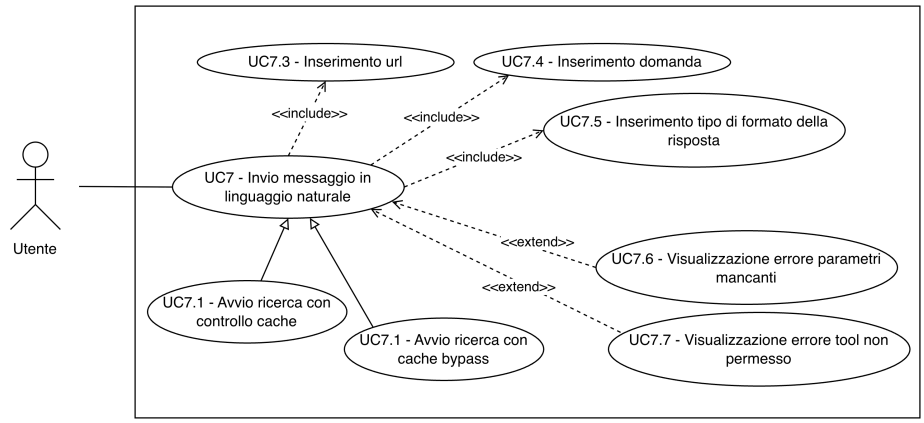

Figura 3.6: Diagramma UC7

Attori	Utente, LLM
Pre-condizioni	• Il sistema è attivo; • l'utente è autenticato nel sistema; • l'utente si trova nella pagina relativa alla *chat*.
Post-condizioni	Il sistema riceve il messaggio dell'utente, comprende il tool richiesto, estrae i parametri necessari alla sua esecuzione e l'LLM avvia l'elaborazione.

Scenario principale	1. L'utente definisce, in linguaggio naturale, l'azione da intraprendere (per esempio "*cerca qual è il meteo attuale a Padova nel sito meteo.it. Rispondi in formato Markdown.*");
	2. l'utente definisce, in linguaggio naturale, i parametri di ricerca:
	  • inserimento url (§3.2.23);
	  • inserimento domanda (§3.2.24);
	  • inserimento tipo di formato della risposta (§3.2.25) (opzionale).
	3. in base ai permessi assegnati all'utente, la ricerca può essere effettuata:
	  • con controllo della *cache* per evitare richieste ridondanti (§3.2.21);
	  • senza controllo della *cache* (*cache bypass*) (§3.2.22).
	4. l'utente invia il messaggio.

Inclusioni	• UC7.3 §3.2.23 • UC7.4 §3.2.24 • UC7.5 §3.2.25

Scenari alternativi	• Alcuni parametri obbligatori per l'esecuzione del tool sono mancanti (§3.2.26); • L'azione richiesta dall'utente non è permessa (§3.2.27).
Estensioni	• UC7.6 §3.2.26 • UC7.7 §3.2.27
Generalizzazioni	• UC7.1 §3.2.21 • UC7.2 §3.2.22

3.2.21 UC7.1 - Avvio tool con controllo *cache*

Attori	Utente, LLM
Pre-condizioni	• L'utente si trova nella pagina relativa alla *chat*; • L'utente ha inviato un messaggio richiedendo un tool; • L'utente ha i permessi di utilizzo del tool e i permessi di uso della *cache*.
Post-condizioni	Il sistema avvia l'azione in base al messaggio inviato dall'utente.
Scenario principale	L'LLM avvia l'elaborazione.

3.2.22 UC7.2 - Avvio tool con *cache bypass*

Attori	Utente, LLM
Pre-condizioni	• L'utente si trova nella pagina relativa alla *chat*; • L'utente ha inviato un messaggio richiedendo un tool; • L'utente ha i permessi di utilizzo del tool ma non ha i permessi di uso della *cache*.
Post-condizioni	Il sistema avvia correttamente l'azione con *cache bypass*, ovvero non effettua il controllo della *cache*.
Scenario principale	L'LLM avvia l'elaborazione.

3.2.23 UC7.3 - Inserimento url

Attori	Utente
Pre-condizioni	L'utente sta inviando un messaggio nella pagina di *chat*.
Post-condizioni	Il sistema riceve l'url.
Scenario principale	L'utente inserisce l'url.

3.2.24 UC7.4 - Inserimento domanda

Attori	Utente
Pre-condizioni	L'utente sta inviando un messaggio nella pagina di *chat*.
Post-condizioni	Il sistema riceve la domanda.
Scenario principale	L'utente inserisce la domanda.

3.2.25 UC7.5 - Inserimento tipo di formato della risposta

Attori	Utente
Pre-condizioni	L'utente sta inviando un messaggio nella pagina di *chat*.
Post-condizioni	Il sistema riceve il tipo di formato con cui generare la risposta.
Scenario principale	L'utente inserisce il tipo di formato della risposta.

3.2.26 UC7.6 - Visualizzazione errore parametri mancanti

Attori	Utente
Pre-condizioni	L'utente non ha inserito uno o più parametri obbligatori per l'elaborazione del tool richiesto.
Post-condizioni	Il sistema non avvia il tool
Scenario principale	Il sistema mostra un messaggio di errore e segnala all'utente quali sono i parametri mancanti.

3.2.27 UC7.7 - Visualizzazione errore tool non permesso

Attori	Utente
Pre-condizioni	L'utente non ha i permessi necessari per eseguire l'azione richiesta oppure il tool richiesto non è valido.
Post-condizioni	Il sistema non avvia l'elaborazione.
Scenario principale	Il sistema mostra un messaggio di errore.

3.2.28 UC8 - Visualizzazione lista dei passaggi della ricerca

Attori	Utente, LLM
Pre-condizioni	• Il sistema è attivo; • l'utente ha inviato un messaggio nella pagina relativa alla *chat*; • l'LLM sta elaborando o ha elaborato la richiesta dell'utente; • l'utente ha i permessi per visualizzare i passaggi della ricerca.
Post-condizioni	L'utente visualizza la sequenza di azioni logiche compiute dall'LLM.

Scenario principale	L'utente visualizza in tempo reale i singoli passaggi generati dall'attività dell'LLM (§3.2.29).
Inclusioni	• UC8.1 §3.2.29

3.2.29 UC8.1 - Visualizzazione singolo passaggio della ricerca

Attori	Utente, LLM
Pre-condizioni	• L'utente ha inviato un messaggio nella pagina relativa alla *chat*; • l'utente ha i permessi per visualizzare i passaggi della ricerca; • l'utente sta visualizzando il flusso di ricerca generato dall'LLM.
Post-condizioni	L'utente visualizza un messaggio informativo riguardante l'operazione specifica effettuata dall'LLM in quel momento.
Scenario principale	1. L'utente visualizza una delle seguenti tipologie di passaggi, in caso il tool sia quello di web crawling: • pagina web estratta e analizzata (§3.2.30); • riassunto semantico del contenuto (§3.2.31); • link trovati nella pagina web analizzata (§3.2.32); • link selezionati per continuare il processo di web crawling (§3.2.33); • risposta finale elaborata (§3.2.34). 2. L'utente, per il singolo passaggio, visualizza le seguenti proprietà: • il nome del passaggio (§3.2.35); • la descrizione del passaggio (§3.2.36). 3. Se al termine dell'elaborazione il sistema genera correttamente la risposta, l'utente può salvare la risposta generata (§3.2.38).
Inclusioni	• UC8.2 §3.2.35 • UC8.3 §3.2.36
Scenari alternativi	Interruzione dell'elaborazione per errore interno o di connessione (§3.2.37).
Estensioni	• UC8.4 §3.2.37
Generalizzazioni	• UC8.1.1 §3.2.30 • UC8.1.2 §3.2.31 • UC8.1.3 §3.2.32 • UC8.1.4 §3.2.33 • UC8.1.5 §3.2.34

3.2.30 UC8.1.1 - Visualizzazione passaggio: pagina analizzata

Attori	Utente, LLM
Pre-condizioni	L'LLM sta estraendo il contenuto da una pagina *web*.
Post-condizioni	L'utente visualizza il nome della pagina *web* da cui si sta estraendo il contenuto.
Scenario principale	Il sistema mostra un messaggio informativo relativo all'estrazione del contenuto da una pagina *web*.

3.2.31 UC8.1.2 - Visualizzazione passaggio: riassunto della pagina analizzata

Attori	Utente, LLM
Pre-condizioni	L'LLM ha completato l'estrazione del testo da una pagina *web*.
Post-condizioni	L'utente visualizza la sintesi generata dall'LLM.
Scenario principale	Il sistema mostra il riassunto informativo che l'LLM ha estratto dal link analizzato per determinare la pertinenza alla domanda.

3.2.32 UC8.1.3 - Visualizzazione passaggio: link trovati

Attori	Utente
Pre-condizioni	• Il sistema ha estratto il testo riassuntivo da una pagina *web*; • il sistema non ha ritenuto sufficiente o pertinente il contenuto trovato per poter rispondere in maniera completa alla domanda dell'utente; • il sistema ha completato l'estrazione di tutti i link a cui fa riferimento la pagina *web* per poter continuare la ricerca.
Post-condizioni	L'utente visualizza la lista dei link trovati.
Scenario principale	Il sistema mostra la lista dei link trovati nella pagina *web* analizzata.

3.2.33 UC8.1.4 - Visualizzazione passaggio: link scelti

Attori	Utente, LLM
Pre-condizioni	L'LLM ha completato la scelta dei link pertinenti per la ricerca.
Post-condizioni	L'utente visualizza la lista dei link scelti dall'LLM per continuare la ricerca. Il massimo numero di link scelti corrisponde al parametro "massima ampiezza" definito dal LLM.
Scenario principale	Il sistema mostra la lista dei link scelti.

3.2.34 UC8.1.5 - Visualizzazione passaggio: risposta

Attori	Utente, LLM

Pre-condizioni	• Il sistema ha estratto il testo riassuntivo da una pagina *web*; • il sistema ha ritenuto sufficiente e pertinente il contenuto trovato per poter rispondere all'utente.
Post-condizioni	L'utente visualizza il passaggio di risposta.
Scenario principale	1. L'LLM genera la risposta finale e la mostra all'utente; 2. il sistema termina la ricerca.

3.2.35 UC8.2 - Visualizzazione nome passaggio

Attori	Utente
Pre-condizioni	L'utente sta visualizzando i passaggi della ricerca avviata.
Post-condizioni	L'utente visualizza il nome del passaggio.
Scenario principale	Il sistema mostra il nome relativo al passaggio visualizzato.

3.2.36 UC8.3 - Visualizzazione descrizione passaggio

Attori	Utente
Pre-condizioni	L'utente sta visualizzando i passaggi della ricerca avviata.
Post-condizioni	L'utente visualizza la descrizione del passaggio.
Scenario principale	Il sistema mostra la descrizione relativa al passaggio visualizzato.

3.2.37 UC8.4 - Visualizzazione errore durante la ricerca

Attori	Utente, LLM
Pre-condizioni	L'LLM riscontra un'impossibilità nel proseguire l'elaborazione della risposta.
Post-condizioni	Il sistema notifica il fallimento della ricerca e ne interrompe l'esecuzione.
Scenario principale	Il sistema mostra un messaggio di errore indicando se l'interruzione è dovuta a problemi interni (es. *timeout*, limiti di contesto) o problemi di rete.

3.2.38 UC8.5 - Salva risposta

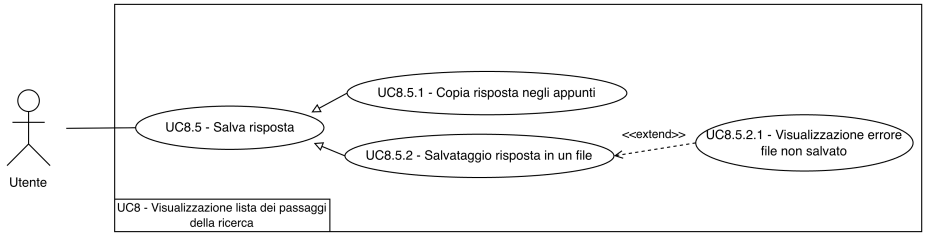

Figura 3.7: Diagramma UC8.5

Attori	Utente
Pre-condizioni	L'utente sta visualizzando la risposta generata correttamente dal sistema.
Post-condizioni	Il contenuto della risposta viene correttamente salvato.
Scenario principale	L'utente seleziona una tra le seguenti opzioni: • Copia la risposta negli appunti (§3.2.39); • Salva in un file la risposta (§3.2.40).
Generalizzazioni	• UC8.5.1 §3.2.39 • UC8.5.2 §3.2.40

3.2.39 UC8.5.1 - Copia risposta negli appunti

Attori	Utente
Pre-condizioni	L'utente sta visualizzando la risposta generata correttamente dal sistema.
Post-condizioni	Il contenuto della risposta viene correttamente copiato negli appunti.
Scenario principale	L'utente seleziona l'opzione di copia negli appunti.

3.2.40 UC8.5.2 - Salvataggio risposta in un file

Attori	Utente
Pre-condizioni	L'utente sta visualizzando la risposta generata correttamente dal sistema.
Post-condizioni	Il contenuto della risposta viene correttamente salvato in un file nel dispositivo dell'utente.
Scenario principale	1. L'utente seleziona l'opzione di creazione di un file in cui salvare la risposta; 2. l'utente sceglie il nome e la posizione in cui salvare il file; 3. l'utente conferma l'operazione.

Scenari alternativi	Il file non viene creato e salvato a causa di un errore durante il procedimento (§3.2.41).
Estensioni	• UC8.5.2.1 §3.2.41

3.2.41 UC8.5.2.1 - Visualizzazione errore file non salvato

Attori	Utente
Pre-condizioni	• L'utente sta visualizzando la risposta generata correttamente dal sistema; • l'utente ha selezionato l'opzione di salvataggio della risposta in un file.
Post-condizioni	Il contenuto della risposta non viene salvato in un file.
Scenario principale	Il sistema mostra all'utente un messaggio di errore.

3.2.42 UC9 - Visualizzazione storico delle chat

Attori	Utente
Pre-condizioni	• Il sistema è attivo; • l'utente è autenticato nel sistema; • l'utente sta visualizzando lo storico delle *chat*.
Post-condizioni	L'utente visualizza la lista di tutte le ricerche nel periodo selezionato.
Scenario principale	1. L'utente seleziona un filtro di periodo, per esempio "*ultima settimana*, *ultimo giorno*, ..." (§3.2.43); 2. l'utente conferma l'operazione; 3. l'utente visualizza i singoli elementi dello storico (§3.2.44).
Inclusioni	• UC9.1 §3.2.43 • UC9.2 §3.2.44 • UC9.2.1 §3.2.45 • UC9.3 §3.2.46

3.2.43 UC9.1 - Selezione filtro periodo

Attori	Utente
Pre-condizioni	L'utente sta visualizzando lo storico delle *chat*.
Post-condizioni	L'utente visualizza lo storico delle *chat* nel periodo selezionato.
Scenario principale	Il sistema mostra la lista di tutte le *chat* nel periodo selezionato dall'utente.

3.2.44 UC9.2 - Visualizzazione singolo elemento dello storico

Attori	Utente
Pre-condizioni	L'utente sta visualizzando lo storico delle *chat*.
Post-condizioni	L'utente visualizza il singolo elemento dello storico.
Scenario principale	Il sistema mostra il singolo elemento dello storico, caratterizzato dall'ultimo messaggio inviato nella *chat*.
Inclusioni	• UC9.2.1 §3.2.45

3.2.45 UC9.2.1 - Visualizzazione ultimo messaggio della chat

Attori	Utente
Pre-condizioni	L'utente sta visualizzando lo storico delle *chat* nel periodo selezionato.
Post-condizioni	L'utente visualizza l'ultimo messaggio della *chat* che può essere un messaggio dell'utente oppure un messaggio del sistema.
Scenario principale	Il sistema mostra parte dell'ultimo messaggio della *chat* nello storico.

3.2.46 UC9.3 - Apertura conversazione dallo storico

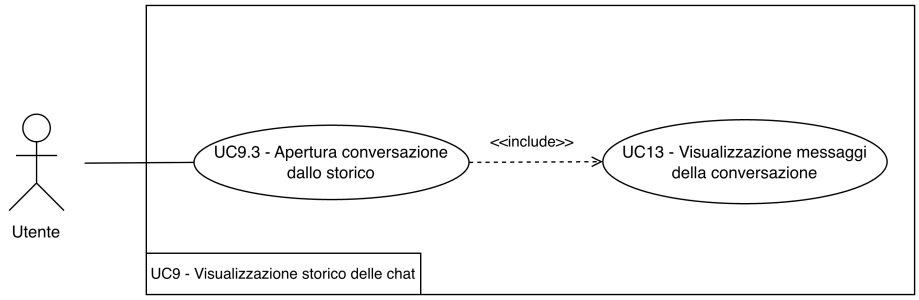

Figura 3.8: Diagramma UC9.3

Attori	Utente
Pre-condizioni	• Il sistema è attivo; • l'utente è autenticato; • l'utente sta visualizzando lo storico delle *chat*; • esiste almeno una conversazione nello storico.
Post-condizioni	L'utente visualizza la conversazione selezionata.
Scenario principale	1. L'utente seleziona una conversazione dallo storico; 2. il sistema recupera tutti i messaggi associati alla conversazione; 3. il sistema mostra il contenuto della conversazione (§3.2.57).

Inclusioni	• UC13 §3.2.57

3.2.47 UC10 - Eliminazione storico

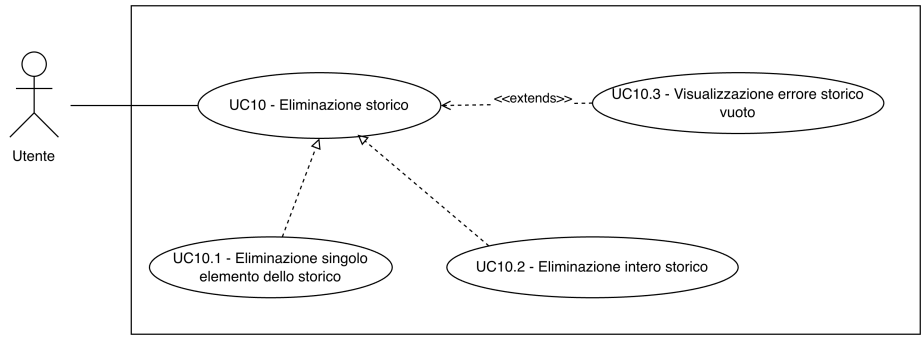

Figura 3.9: Diagramma UC10

Attori	Utente
Pre-condizioni	• Il sistema è attivo; • l'utente è autenticato nel sistema; • l'utente si trova nella pagina di storico delle ricerche; • l'utente ha i permessi per poter visualizzare lo storico delle ricerche.
Post-condizioni	L'utente elimina correttamente le richieste dallo storico.
Scenario principale	L'utente può effettuare una delle seguenti tipologie di eliminazione dallo storico: • Eliminazione del singolo elemento dello storico (§3.2.48); • eliminazione di tutto lo storico (§3.2.49).
Scenari alternativi	Lo storico è vuoto (§3.2.50).
Estensioni	• UC10.3 §3.2.50
Generalizzazioni	• UC10.1 §3.2.48 • UC10.2 §3.2.49

3.2.48 UC10.1 - Eliminazione singolo elemento dello storico

Attori	Utente
Pre-condizioni	L'utente sta visualizzando lo storico delle *chat* nel periodo selezionato.
Post-condizioni	L'utente elimina correttamente il singolo elemento dello storico selezionato.

Scenario principale	1. L'utente seleziona l'elemento che vuole eliminare; 2. l'utente seleziona l'opzione relativa all'eliminazione dell'elemento; 3. l'utente conferma l'operazione.

3.2.49 UC10.2 - Eliminazione intero storico

Attori	Utente
Pre-condizioni	L'utente sta visualizzando lo storico delle *chat* nel periodo selezionato.
Post-condizioni	L'utente elimina correttamente tutto lo storico, indipendentemente dal periodo selezionato.
Scenario principale	1. l'utente seleziona l'opzione relativa all'eliminazione di tutto lo storico, indipendentemente dal periodo selezionato; 2. l'utente conferma l'operazione.

3.2.50 UC10.3 - Visualizzazione errore storico vuoto

Attori	Utente
Pre-condizioni	L'utente ha selezionato l'opzione di eliminazione dello storico delle richieste che è vuoto.
Post-condizioni	Il sistema non elimina lo storico.
Scenario principale	Il sistema mostra un messaggio di errore.

3.2.51 UC11 - Visualizzazione lista utenti

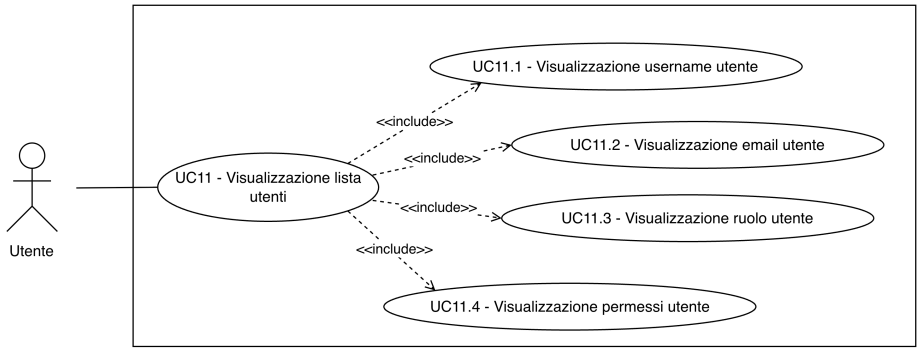

Figura 3.10: Diagramma UC11

Attori	Admin

Pre-condizioni	• Il sistema è attivo; • l'admin è autenticato nel sistema; • l'admin si trova nella pagina di gestione degli utenti.
Post-condizioni	L'admin visualizza la lista degli utenti registrati nel sistema.

Scenario principale	1. L'admin accede alla sezione di gestione utenti;
	2. il sistema recupera la lista degli utenti registrati;
	3. l'admin visualizza le informazioni associate a ciascun utente:
	  • username (§3.2.52);
	  • email (§3.2.53);
	  • ruolo (§3.2.54);
	  • permessi (§3.2.55).

Inclusioni	• UC11.1 §3.2.52 • UC11.2 §3.2.53 • UC11.3 §3.2.54 • UC11.4 §3.2.55

3.2.52 UC11.1 - Visualizzazione username utente

Attori	Admin
Pre-condizioni	L'admin sta visualizzando la lista utenti.
Post-condizioni	L'admin visualizza lo username dell'utente.
Scenario principale	Il sistema mostra lo username associato all'utente.

3.2.53 UC11.2 - Visualizzazione email utente

Attori	Admin
Pre-condizioni	L'admin sta visualizzando la lista utenti.
Post-condizioni	L'admin visualizza l'email dell'utente.
Scenario principale	Il sistema mostra l'email associata all'utente.

3.2.54 UC11.3 - Visualizzazione ruolo utente

Attori	Admin
Pre-condizioni	L'admin sta visualizzando la lista utenti.
Post-condizioni	L'admin visualizza il ruolo dell'utente.
Scenario principale	Il sistema mostra il ruolo associato all'utente.

3.2.55 UC11.4 - Visualizzazione permessi utente

Attori	Admin
Pre-condizioni	L'admin sta visualizzando la lista utenti.
Post-condizioni	L'admin visualizza i permessi dell'utente.
Scenario principale	Il sistema mostra i permessi associati all'utente.

3.2.56 UC12 - Creazione nuova chat

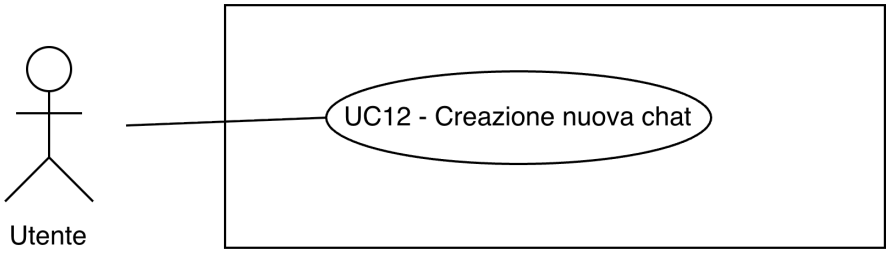

Figura 3.11: Diagramma UC12

Attori	Utente
Pre-condizioni	• Il sistema è attivo; • l'utente è autenticato.
Post-condizioni	Il sistema crea una nuova conversazione vuota.
Scenario principale	1. L'utente seleziona l'opzione di creazione di una nuova chat; 2. il sistema genera una nuova conversazione; 3. il sistema mostra la nuova chat pronta a ricevere messaggi.

3.2.57 UC13 - Visualizzazione messaggi della chat

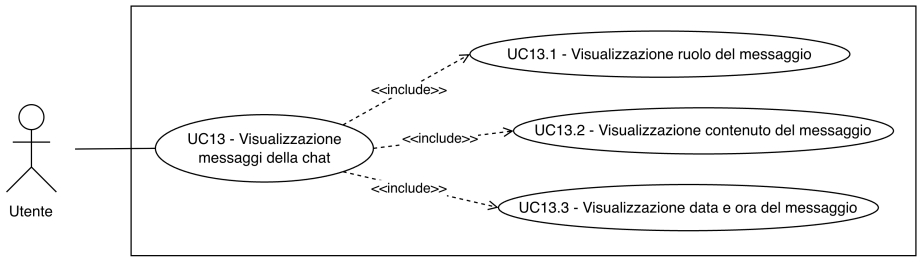

Figura 3.12: Diagramma UC13

Attori	Utente
Pre-condizioni	• Il sistema è attivo; • l'utente è autenticato; • l'utente ha selezionato una *chat*.
Post-condizioni	L'utente visualizza i messaggi della conversazione selezionata.

Scenario principale	1. L'utente seleziona una *chat*;
	2. il sistema recupera tutti i messaggi associati;
	3. il sistema mostra i messaggi della conversazione:
	  • ruolo del messaggio (§3.2.58);
	  • contenuto del messaggio (§3.2.59);
	  • data e ora del messaggio (§3.2.60).

Inclusioni	• UC13.1 §3.2.58 • UC13.2 §3.2.59 • UC13.3 §3.2.60

3.2.58 UC13.1 - Visualizzazione ruolo del messaggio

Attori	Utente
Pre-condizioni	L'utente sta visualizzando una *chat*.
Post-condizioni	L'utente visualizza il ruolo associato al messaggio.
Scenario principale	Il sistema mostra il ruolo del messaggio (utente o sistema).

3.2.59 UC13.2 - Visualizzazione contenuto del messaggio

Attori	Utente
Pre-condizioni	L'utente sta visualizzando una *chat*.
Post-condizioni	L'utente visualizza il contenuto del messaggio.
Scenario principale	Il sistema mostra il contenuto testuale del messaggio.

3.2.60 UC13.3 - Visualizzazione data e ora del messaggio

Attori	Utente
Pre-condizioni	L'utente sta visualizzando una *chat*.
Post-condizioni	L'utente visualizza la data e l'ora del messaggio.
Scenario principale	Il sistema mostra la data e l'ora di invio del messaggio.

3.3 Requisiti

Da un'attenta analisi dei casi d'uso svolta è stata stilata la tabella per il tracciamento dei requisiti in relazione agli *usecase* definiti. Sono stati individuati vari tipi di requisiti e si è quindi fatto uso di un codice identificativo per distinguerli:

- **Funzionali** (RF): descrivono le funzionalità che un sistema deve avere per soddisfare i bisogni. Si suddividono in requisiti funzionali obbligatori, opzionali e desiderabili;
- **Di qualità** (RQ): requisiti che devono essere rispettati per accertare la qualità della progettazione e dello sviluppo del sistema;
- **Di vincolo** (RV): restrizioni che devono essere rispettate durante la progettazione e lo sviluppo del sistema.

3.3.1 Funzionali

ID	Tipo	Descrizione	Fonti
RF-1	Obbligatorio	L'utente deve potersi autenticare presso il sistema	UC1 §3.2.1
RF-2	Obbligatorio	L'utente deve inserire il proprio *username* per autenticarsi presso il sistema	UC1.1 §3.2.2
RF-3	Obbligatorio	L'utente deve inserire la propria password per autenticarsi presso il sistema	UC1.2 §3.2.3
RF-4	Obbligatorio	L'utente deve ricevere un errore se le credenziali inserite sono errate	UC1.3 §3.2.4
RF-5	Obbligatorio	L'utente deve poter effettuare il *logout* dal sistema	UC2 §3.2.5
RF-6	Obbligatorio	L'admin deve poter creare un nuovo utente nel sistema	UC3 §3.2.6
RF-7	Obbligatorio	L'admin deve poter inserire una email per il nuovo utente	UC3.1 §3.2.7
RF-8	Obbligatorio	L'admin deve poter inserire uno *username* per il nuovo utente	UC3.2 §3.2.8
RF-9	Obbligatorio	L'admin deve poter inserire una password per il nuovo utente	UC3.3 §3.2.9
RF-10	Obbligatorio	L'admin deve poter selezionare il ruolo del nuovo utente	UC3.4 §3.2.10
RF-11	Obbligatorio	L'admin deve ricevere un errore se la email inserita è già esistente nel sistema	UC3.5 §3.2.11
RF-12	Obbligatorio	L'admin deve ricevere un errore se lo *username* inserito è già esistente nel sistema	UC3.6 §3.2.12
RF-13	Obbligatorio	L'admin deve poter eliminare un utente dal sistema	UC4 §3.2.13
RF-14	Obbligatorio	L'utente deve poter visualizzare il proprio profilo	UC5 §3.2.14
RF-15	Obbligatorio	L'utente deve poter visualizzare il proprio *username* nel profilo	UC5.1 §3.2.15
RF-16	Obbligatorio	L'utente deve poter visualizzare la propria email nel profilo	UC5.2 §3.2.16
RF-17	Obbligatorio	L'utente deve poter modificare il proprio profilo	UC6 §3.2.17

RF-18	Obbligatorio	L'utente deve poter modificare il proprio *username*	UC6.1 §3.2.18
RF-19	Obbligatorio	L'utente deve poter modificare la propria password	UC6.2 §3.2.19
RF-20	Obbligatorio	L'utente deve poter inviare un messaggio in linguaggio naturale per avviare una ricerca	UC7 §3.2.20
RF-21	Obbligatorio	L'utente deve poter avviare una ricerca con *cache bypass*	UC7.2 §3.2.22
RF-22	Obbligatorio	Il sistema deve poter estrarre dal messaggio dell'utente i parametri necessari all'esecuzione del tool richiesto	UC7.3 §3.2.23
RF-23	Obbligatorio	Il sistema deve estrarre la domanda dal messaggio dell'utente per avviare la ricerca	UC7.4 §3.2.24
RF-24	Obbligatorio	Il sistema deve segnalare all'utente i parametri obbligatori mancanti, impedendo l'avvio del tool	UC7.6 §3.2.26
RF-25	Obbligatorio	L'utente deve ricevere un errore se l'azione richiesta non è permessa o il tool richiesto non è valido	UC7.7 §3.2.27
RF-26	Obbligatorio	L'utente deve poter visualizzare la risposta generata dal sistema	UC8.1.5 §3.2.34
RF-27	Obbligatorio	Il sistema deve notificare l'utente in caso di errore durante la ricerca	UC8.4 §3.2.37
RF-28	Obbligatorio	L'admin deve poter visualizzare la lista degli utenti registrati nel sistema	UC11 §3.2.51
RF-29	Obbligatorio	L'admin deve poter visualizzare lo username degli utenti	UC11.1 §3.2.52
RF-30	Obbligatorio	L'admin deve poter visualizzare l'email degli utenti	UC11.2 §3.2.53
RF-31	Obbligatorio	L'admin deve poter visualizzare il ruolo degli utenti	UC11.3 §3.2.54
RF-32	Obbligatorio	L'admin deve poter visualizzare i permessi degli utenti	UC11.4 §3.2.55
RF-33	Obbligatorio	L'utente deve poter creare una nuova conversazione	UC12 §3.2.56
RF-34	Opzionale	L'utente deve poter specificare il formato della risposta (Markdown, HTML, JSON)	UC7.5 §3.2.25
RF-35	Opzionale	L'utente deve poter visualizzare in tempo reale i passaggi della ricerca in corso	UC8 §3.2.28
RF-36	Opzionale	L'utente deve poter visualizzare in tempo reale il singolo passaggio della ricerca in corso	UC8.1 §3.2.29
RF-37	Opzionale	L'utente deve poter visualizzare il passaggio relativo alla pagina *web* in fase di analisi	UC8.1.1 §3.2.30
RF-38	Opzionale	L'utente deve poter visualizzare il passaggio relativo al riassunto semantico della pagina analizzata	UC8.1.2 §3.2.31
RF-39	Opzionale	L'utente deve poter visualizzare il passaggio relativo ai link trovati nella pagina analizzata	UC8.1.3 §3.2.32

RF-40	Opzionale	L'utente deve poter visualizzare il passaggio relativo ai link scelti per continuare la ricerca	UC8.1.4 §3.2.33
RF-41	Opzionale	L'utente deve poter visualizzare il nome del passaggio	UC8.2 §3.2.35
RF-42	Opzionale	L'utente deve poter visualizzare la descrizione del passaggio	UC8.3 §3.2.36
RF-43	Opzionale	L'utente deve poter visualizzare lo storico delle proprie *chat*	UC9 §3.2.42
RF-44	Opzionale	L'utente deve poter filtrare lo storico per periodo temporale (ultima settimana, ultimo giorno, ...)	UC9.1 §3.2.43
RF-45	Opzionale	L'utente deve poter visualizzare i singoli elementi dello storico	UC9.2 §3.2.44
RF-46	Opzionale	L'utente deve poter visualizzare, per ogni elemento dello storico, l'ultimo messaggio della relativa *chat*	UC9.2.1 §3.2.45
RF-47	Opzionale	L'utente deve poter aprire una conversazione presente nello storico	UC9.3 §3.2.46
RF-48	Opzionale	L'utente deve poter eliminare lo storico delle proprie *chat*	UC10 §3.2.47
RF-49	Opzionale	L'utente deve poter eliminare il singolo elemento dello storico	UC10.1 §3.2.48
RF-50	Opzionale	L'utente deve poter eliminare tutto lo storico	UC10.2 §3.2.49
RF-51	Opzionale	L'utente deve ricevere un errore se tenta di eliminare lo storico quando è vuoto	UC10.3 §3.2.50
RF-52	Opzionale	L'utente deve poter visualizzare i messaggi di una conversazione	UC13 §3.2.57
RF-53	Opzionale	L'utente deve poter visualizzare il ruolo associato a un messaggio	UC13.1 §3.2.58
RF-54	Opzionale	L'utente deve poter visualizzare il contenuto di un messaggio	UC13.2 §3.2.59
RF-55	Opzionale	L'utente deve poter visualizzare la data e l'ora di invio di un messaggio	UC13.3 §3.2.60
RF-56	Desiderabile	L'utente deve poter avviare una ricerca con controllo della *cache*	UC7.1 §3.2.21
RF-57	Desiderabile	L'utente deve poter salvare la risposta generata dal sistema	UC8.5 §3.2.38
RF-58	Desiderabile	L'utente deve poter copiare la risposta generata negli appunti	UC8.5.1 §3.2.39
RF-59	Desiderabile	L'utente deve poter salvare la risposta generata in un file locale	UC8.5.2 §3.2.40
RF-60	Desiderabile	L'utente deve ricevere un errore se il file con la risposta generata non viene salvato correttamente	UC8.5.2.1 §3.2.41

Tabella 3.63: Requisiti funzionali

3.3.2 Di qualità

ID	Tipo	Descrizione	Fonti
RQ-1	Obbligatorio	Il sistema deve essere corredato da almeno un test di unità che verifichi il corretto funzionamento del *routing* dei *MCP tool*	Intervista
RQ-2	Obbligatorio	Il sistema deve essere corredato da almeno un test per ogni funzionalità fornita dai *MCP tool*	Intervista
RQ-3	Obbligatorio	Il codice sorgente deve essere contenuto in un *repository* versionato tramite *Git*	Intervista
RQ-4	Obbligatorio	Consegnare la documentazione dei casi d'uso per l'utilizzo dell'applicativo *web*	Piano di lavoro
RQ-5	Obbligatorio	Consegnare la specifica tecnica dell'applicativo *web*	Piano di lavoro
RQ-6	Obbligatorio	Consegnare il manuale di utilizzo dell'applicativo *web* per gli utenti	Piano di lavoro

Tabella 3.64: Requisiti di qualità

3.3.3 Di vincolo

ID	Tipo	Descrizione	Fonti
RV-1	Obbligatorio	Il sistema deve essere containerizzato tramite Docker	Piano di lavoro
RV-2	Obbligatorio	Il sistema deve utilizzare JWT per l'autenticazione degli utenti.	Intervista
RV-3	Obbligatorio	Il *frontend* deve essere sviluppato usando il framework React	Intervista
RV-4	Obbligatorio	Il *backend* deve essere sviluppato usando il framework FastAPI	Piano di lavoro
RV-5	Obbligatorio	Le componenti del sistema devono comunicare tramite API	Piano di lavoro

Tabella 3.65: Requisiti di vincolo

3.3.4 Riepilogo

Tipi di requisiti	Obbligatori	Opzionali	Desiderabili	Totali
Funzionali	33	22	5	60
Di qualità	6	0	0	6
Di vincolo	5	0	0	5
Totali	44	22	5	**71**

Tabella 3.66: Riepilogo dei requisiti

Capitolo 4

Progettazione e codifica

In questo capitolo viene descritta la progettazione del sistema e le tecnologie utilizzate per lo sviluppo del prodotto software.

4.1 Tecnologie e strumenti

Il prodotto software sviluppato ha l'obiettivo di consentire agli utenti di interagire con un *chatbot* capace di selezionare, in base alla richiesta formulata, il tool più appropriato tra quelli esposti dal server MCP, inclusa la funzionalità di web crawling implementata in questo progetto.

La realizzazione di tale sistema richiede una comprensione approfondita delle tecniche di *caching* semantico e delle tecnologie di intelligenza artificiale applicate al recupero e all'elaborazione delle informazioni testuali.

Le tecnologie destinate al progetto sono state selezionate con l'obiettivo di rispondere, nella maniera più efficiente possibile, alle seguenti sfide architetturali e funzionali:

- **Gestione dei dati e autenticazione:** amministrazione sicura delle informazioni e profilazione degli utenti;
- **Ottimizzazione del *crawling*:** ottimizzazione del processo di estrazione dei contenuti dalle pagine web;
- **Caching semantico:** progettazione di un meccanismo di memorizzazione intermedia per ridurre le richieste ridondanti all'LLM, ottimizzando tempi e costi;
- **Interfaccia utente reattiva:** sviluppo di una *UI* dinamica, capace di mostrare in tempo reale i singoli passaggi del processo di ricerca;
- **Integrazione dell'ecosistema MCP:** implementazione di un server MCP che esponga il servizio di web crawling come strumento standardizzato, rendendolo fluidamente selezionabile dal *chatbot* tra i vari tool disponibili.

4.1.1 Strumenti utilizzati

In questa sezione vengono discusse le principali tecnologie individuate durante le prime settimane di tirocinio e utilizzate nello sviluppo del progetto.

React

React è una libreria JavaScript *open source* ampiamente utilizzata per la costruzione di interfacce utente dichiarative e basate su componenti. La sua adozione in questo progetto riflette la necessità di costruire un'interfaccia complessa, reattiva e facilmente manutenibile per la gestione delle conversazioni e degli utenti. React si basa sul concetto di componente: ogni elemento dell'interfaccia viene scomposto in unità indipendenti, riutilizzabili e componibili tra loro. Questo approccio facilita la suddivisione del lavoro di sviluppo e la manutenzione del codice nel tempo, poiché ogni componente può essere modificato o testato in isolamento senza impattare il resto dell'applicazione.

Un altro aspetto rilevante di React è il suo *Virtual DOM*, un meccanismo che consente di aggiornare dinamicamente gli oggetti nelle pagine, evitando operazioni di *refresh* dell'intera pagina. Questa caratteristica risulta particolarmente utile in questo progetto, dove l'interfaccia deve aggiornarsi di continuo per mostrare in tempo reale i singoli passaggi durante l'elaborazione della risposta da parte del LLM. React vanta inoltre un insieme vasto di librerie di supporto, tra cui *React Router* per la gestione della navigazione e *i18next* per l'internazionalizzazione dei contenuti, entrambe impiegate in questo progetto.

Per questi motivi, React rappresenta una scelta solida ed efficace per lo sviluppo del *frontend* di questa applicazione.

Vite

Vite è uno strumento di compilazione per applicazioni web moderne, progettato per offrire un'esperienza di sviluppo estremamente rapida. Infatti, Vite sfrutta i moduli *ES* (*ECMAScript Modules*) nativi del *browser* durante la fase di sviluppo, evitando di dover ricompilare l'intera applicazione ad ogni modifica del codice. Questo strumento si distingue per i tempi di avvio quasi istantanei e permette di visualizzare le modifiche al codice nel *browser* in tempi minimi, senza dover ricaricare l'intera pagina.

La scelta di Vite integrata con React è motivata dalla necessità di mantenere un flusso di sviluppo rapido ed efficiente durante l'iterazione continua sui componenti dell'interfaccia.

Python

Python è un linguaggio di programmazione ad alto livello, interpretato e di uso generale, noto per la sua sintassi chiara e leggibile. Tale caratteristica favorisce la leggibilità del codice e riduce i costi di manutenzione del progetto, aspetto particolarmente rilevante in un'architettura suddivisa in molti microservizi indipendenti. Python si distingue per la sua versatilità: può essere impiegato per lo sviluppo *backend*, l'elaborazione di dati, l'integrazione con modelli di intelligenza artificiale e l'automazione di processi. Questa flessibilità lo rende particolarmente adatto a un progetto come questo, in cui il linguaggio deve gestire sia la logica di business dei vari servizi sia l'orchestrazione delle chiamate al LLM.

Per questi motivi, Python rappresenta la scelta naturale per l'implementazione di tutti i servizi *backend* del sistema.

FastAPI

FastAPI è un framework web moderno per Python, progettato per la costruzione di API ad alte prestazioni. Si basa sugli standard *OpenAI* e *JSON Schema*, e sfrutta i *type hints* di Python per la validazione automatica dei dati in ingresso e in uscita. Una delle caratteristiche principali di FastAPI è la generazione automatica della documentazione interattiva delle API, basata sugli schemi *Pydantic* definiti nel codice. Questo riduce sensibilmente il tempo necessario per la redazione della specifica degli *endpoint* e garantisce che quest'ultima sia sempre sincronizzata con l'implementazione effettiva. FastAPI è inoltre costruito su *Starlette* e supporta nativamente la programmazione asincrona, aspetto fondamentale per questo progetto, dove i singoli microservizi devono gestire un elevato numero di richieste e comunicazioni in tempo reale tramite SSE.

Per queste ragioni, FastAPI è stato scelto come framework principale per l'implementazione dei servizi *backend* del sistema, secondo un'architettura a porte e adattatori (*hexagonal architecture*).

FastMCP e il Model Context Protocol

Il MCP è un protocollo standardizzato che consente ai modelli di intelligenza artificiale di interagire con strumenti e servizi esterni tramite un'interfaccia uniforme, eliminando la necessità di implementare integrazioni specifiche per ciascun provider o strumento. FastMCP è un framework Python che semplifica notevolmente l'implementazione di un server MCP, occupandosi della gestione del protocollo, della definizione degli strumenti esposti e della comunicazione con i client. FastMCP si distingue per la sua semplicità d'uso: permette di esporre funzioni Python come strumenti utilizzabili da un modello di linguaggio tramite semplici decoratori (come `@mcp.tool`), senza richiedere la gestione manuale dei dettagli di basso livello del protocollo.

Nel contesto di questo progetto, FastMCP rappresenta il cuore del sistema: il server espone, tra i propri strumenti, il tool di web crawling, che il *chatbot* è in grado di selezionare automaticamente quando il messaggio dell'utente lo richiede. La scelta di FastMCP consente quindi di concentrarsi sulla logica di estrazione e ricerca dei contenuti, delegando a tale strumento la gestione del protocollo di comunicazione e di esposizione degli strumenti al modello.

BeautifulSoup

BeautifulSoup è una libreria Python per l'estrazione del contenuto da documenti HTML e XML[g]. Questa libreria si distingue per la sua tolleranza nei confronti di HTML malformato, una caratteristica particolarmente utile quando si analizzano pagine web reali, che spesso non rispettano rigorosamente gli standard. BeautifulSoup fornisce inoltre un insieme di metodi per la ricerca e il filtraggio degli elementi della pagina in base a *tag*, classi, identificatori o altri attributi, semplificando notevolmente l'estrazione delle informazioni rilevanti.

Nel contesto di questo progetto, BeautifulSoup viene impiegata nella fase di recupero delle pagine web permettendo di prelevare il loro contenuto e i collegamenti ipertestuali, necessari per la generazione dei riassunti e la scelta dei *link* per il processo di web crawling.

OpenAI

OpenAI è un'organizzazione di ricerca e sviluppo nel campo dell'intelligenza artificiale, celebre per i modelli GPT (*Generative Pre-trained Transformer*). I modelli offerti da OpenAI sono accessibili tramite servizi *cloud* e possono essere utilizzati attraverso API, permettendo di realizzare sistemi basati sulla comprensione del linguaggio naturale.

Nel contesto di questo progetto, OpenAI rappresenta il componente centrale per l'elaborazione delle richieste degli utenti e la generazione delle risposte.

LangSmith

LangSmith è una piattaforma di tracciabilità e *debugging* pensata specificamente per applicazioni basate su modelli di intelligenza artificiale. Permette di tracciare in modo dettagliato l'esecuzione delle catene di chiamate che coinvolgono LLM, fornendo una visione completa del flusso di esecuzione. Questa piattaforma si distingue per la sua capacità di raccogliere *trace* distribuite anche in architetture a microservizi, dove una singola richiesta dell'utente può attraversare più servizi prima di giungere a una risposta finale. Ogni passaggio viene registrato con i relativi *input*, *output*, tempi di esecuzione e metadati, permettendo di individuare rapidamente latenze, errori o comportamenti inattesi del modello.

Nel contesto di questo progetto, LangSmith è stato adottato per il tracciamento distribuito delle operazioni svolte durante il processo di web crawling, dal momento della selezione del tool da parte del *chatbot* fino alla generazione della risposta finale. Questo strumento si è rivelato fondamentale durante le fasi di sviluppo e *debug* del sistema, permettendo di analizzare nel dettaglio il comportamento del modello.

PostgreSQL, TimescaleDB e pgvector

PostgreSQL è un sistema di gestione di basi di dati relazionali *open source*, noto per la sua robustezza, l'affidabilità e la conformità agli standard SQL[g]. La sua espandibilità tramite estensioni native lo rende particolarmente adatto a progetti che richiedono funzionalità avanzate oltre alla semplice gestione di dati relazionali, come nel caso di questo progetto. TimescaleDB è un'estensione di PostgreSQL pensata per la gestione efficiente di dati temporali (*time-series*), ottimizzata per l'inserimento e l'interrogazione di grandi volumi di dati ordinati nel tempo. Tale estensione è stata adottata per la gestione dello storico delle ricerche effettuate dagli utenti, dove ogni elemento è naturalmente associato a una dimensione temporale. L'estensione pgvector introduce invece un tipo di dato vettoriale e operatori di similarità, permettendo di memorizzare e interrogare *embedding* direttamente all'interno del database relazionale. Questa estensione è fondamentale per l'implementazione del meccanismo di *caching* semantico del sistema: le domande poste dagli utenti vengono trasformate in vettori numerici e confrontate con quelle già elaborate in precedenza, individuando richieste semanticamente simili ed evitando così rielaborazioni ridondanti da parte dell'LLM.

La combinazione di queste tre tecnologie permette di utilizzare un unico database per gestire sia i dati relazionali tradizionali (utenti, storico delle conversazioni, profili) sia i dati vettoriali necessari al *caching* semantico, semplificando l'infrastruttura complessiva del sistema.

Docker e Docker Compose

Docker è una piattaforma di containerizzazione che consente di raggruppare un'applicazione e tutte le sue dipendenze in un'unità isolata e portabile, denominata "container". Questo approccio garantisce che l'applicazione si comporti nello stesso modo indipendentemente dall'ambiente in cui viene eseguita, eliminando i classici problemi di incompatibilità tra ambienti di sviluppo, test e produzione. Docker Compose è uno strumento che permette di definire e orchestrare applicazioni multi-container tramite un singolo file di configurazione dichiarativo.

Questo strumento risulta particolarmente utile in questo progetto dove l'architettura del sistema è suddivisa in molteplici microservizi indipendenti e si richiede scalabilità e portabilità.

Nginx

Nginx è un *reverse proxy open source*, noto per le sue elevate prestazioni e per la capacità di gestire un numero elevato di connessioni simultanee con un basso utilizzo di risorse. In questo progetto, Nginx svolge il ruolo di *gateway* per l'intera applicazione, instradando le richieste provenienti dal *frontend* verso i corretti microservizi *backend*. Questo approccio centralizza la gestione del traffico in ingresso, semplificando la configurazione dell'instradamento delle richieste e permettendo di disaccoppiare il *frontend* dalla conoscenza della struttura interna del sistema a microservizi.

La scelta di Nginx come *gateway* riflette quindi l'esigenza di un punto di ingresso unico, performante e configurabile per l'intera architettura distribuita del sistema.

JWT e JWKS

JWT è uno standard aperto per la creazione di *token* di accesso che permettono di trasmettere informazioni in modo sicuro tra due parti, sotto forma di oggetto JSON firmato digitalmente. Nel contesto di questo progetto, i JWT vengono utilizzati per gestire l'autenticazione degli utenti tra i diversi microservizi del sistema.

In particolare, è stato adottato l'algoritmo di firma *RS256*, basato su una coppia di chiavi asimmetriche: una chiave privata, utilizzata dal servizio responsabile dell'autenticazione, per firmare i *token* al momento del *login*, e una chiave pubblica, distribuita agli altri servizi tramite JWKS[g], per la verifica della validità dei *token*.

L'adozione di *RS256* e JWKS risponde quindi a un'esigenza di sicurezza e di efficienza architetturale, garantendo che ogni servizio possa operare in autonomia mantenendo al contempo un meccanismo di autenticazione robusto e centralizzato nella sua gestione delle chiavi.

4.2 Progettazione

L'infrastruttura è stata definita secondo un modello stratificato (*multitier*), strutturato su due livelli macroscopici: il livello di presentazione (*frontend*) e lo strato di logica applicativa (*backend*).

4.2.1 Progettazione backend

Il *backend* rappresenta il nucleo computazionale dell'applicazione. Viene suddiviso secondo l'architettura a microservizi, una scelta mirata a garantire un elevato livello di modularità e di separazione delle responsabilità.

L'intero livello è sviluppato in linguaggio Python sfruttando i framework FastAPI e FastMCP. A seguire vengono descritti nel dettaglio i singoli microservizi e le principali sfide affrontate nel loro sviluppo.

MCP Server Service

Il MCP Server Service si basa sul protocollo MCP ed è responsabile dell'esposizione del tool di *crawling*. Esso si occupa, inoltre, di generare gli eventi SSE e inviarli verso il *frontend* per notificare all'utente, in tempo reale, i singoli passaggi logici del processo.

La difficoltà maggiore durante la codifica di tale servizio ha riguardato l'ingegnerizzazione della logica di ricorsione del processo di *crawling*. Ad ogni passo ricorsivo, l'algoritmo esegue in modo sequenziale i seguenti passaggi:

1. **Estrazione:** recupero del contenuto HTML della pagina corrente X;
2. **Sintesi:** generazione di un riassunto mirato in base alla richiesta originaria dell'utente, il quale viene accodato alla lista dei riassunti raccolti nei passi precedenti;
3. **Valutazione del contesto:** verifica della sufficienza informativa dei dati raccolti per la generazione della risposta, effettuata analizzando l'intera lista dei riassunti accumulati. Tale controllo determina due possibili scenari:
   - **Successo**: il contesto è valutato come esaustivo per rispondere al quesito; la ricorsione viene interrotta e si procede alla generazione della risposta finale;
   - **Prosecuzione**: i riassunti correnti sono considerati insufficienti per formulare una risposta completa, rendendo necessaria la continuazione del processo di *crawl*;
4. **Parsing dei collegamenti:** in caso di mancata risoluzione al punto precedente, vengono estratti dalla pagina X tutti i collegamenti ipertestuali disponibili;
5. **Filtraggio semantico:** selezione dei *link* ritenuti maggiormente pertinenti rispetto alla domanda dell'utente;
6. **Passo ricorsivo:** per ciascun collegamento selezionato, viene avviata una nuova istanza della medesima funzione ricorsiva.

La logica sopra descritta modella l'esplorazione secondo una struttura ad albero. In questa analogia algoritmica, l'**ampiezza** della struttura è determinata dal fattore di *branching*, ossia dal numero massimo di collegamenti selezionati durante la fase di filtraggio semantico (passo 5); l'**altezza** dell'albero (ovvero la sua profondità massima) corrisponde invece al limite massimo di chiamate ricorsive nidificate lungo il cammino più lungo che intercorre tra il nodo radice e un nodo foglia.

Complessità ricorsiva La principale criticità dell'algoritmo descritto risiede nella definizione dei parametri di ampiezza e profondità. La configurazione di tali valori richiede una calibrazione estremamente rigorosa: una scelta sovradimensionata esporrebbe il sistema al fenomeno dell'*esplosione combinatoria*. Ciò comporterebbe un consumo eccessivo di risorse computazionali e un incremento incontrollato delle chiamate verso l'LLM, con conseguenti ripercussioni in termini di latenza e costi finanziari. Di contro, l'adozione di vincoli stringenti comprometterebbe l'efficacia stessa del processo di ricerca: in caso di domande più complesse, dei parametri troppo bassi condurrebbero inevitabilmente a un'analisi superficiale e, di conseguenza, a un fallimento nel soddisfare la richiesta dell'utente.

La Tabella 4.1 mostra come i parametri di ampiezza e profondità influenzano la complessità ricorsiva nel caso peggiore (ovvero qualora l'algoritmo esaurisse l'intero albero senza raggiungere prima lo stato di successo). Nel processo di *crawling*, ogni singolo passo (nodo dell'albero) richiede esattamente tre chiamate distinte all'LLM (fasi di sintesi, valutazione del contesto e selezione dei *link*).

Caso	Ampiezza	Profondità	Chiamate di ricorsione	Chiamate al LLM
1	3	6	1092	3276
2	1	4	4	12
3	2	6	126	378
4	2	5	62	186

Tabella 4.1: Analisi dell'impatto dei parametri di profondità e ampiezza sul numero di chiamate.

Sulla base delle analisi effettuate durante la fase di *testing* e bilanciando il *trade-off* tra l'accuratezza del reperimento informativo (*recall*) e il contenimento di costi e latenze, la configurazione ottimale è stata individuata nei seguenti valori:

- **Ampiezza = 2**: Limitare il fattore di *branching* a un massimo di due collegamenti ipertestuali per pagina previene l'espansione iperbolica dell'albero;
- **Profondità = 5**: Configurare l'altezza massima a cinque livelli di nidificazione consente all'algoritmo di penetrare sufficientemente a fondo nella struttura del sito web.

Con questa specifica impostazione ($b = 2, d = 5$), il numero massimo teorico di nodi esplorabili è pari a $N_{\max} = 2^1 + 2^2 + 2^3 + 2^4 + 2^5 = 62$. Di conseguenza, sempre nel caso peggiore, il numero massimo di chiamate all'LLM si attesta a:

$$C_{tot} = 62 \times 3 = 186 \text{ chiamate} \tag{4.1}$$

Tale valore rappresenta un limite sostenibile per l'infrastruttura e perfettamente in linea con le capacità operative dei modelli di intelligenza artificiale aziendali.

Collegamenti ipertestuali ciclici Un'ulteriore criticità strutturale è rappresentata dalla presenza di collegamenti ipertestuali ciclici (o *link* circolari) all'interno delle pagine web esplorate. In assenza di un opportuno meccanismo di tracciamento, l'algoritmo rischierebbe di rimanere intrappolato in cicli di navigazione infiniti tra pagine che si referenziano reciprocamente.

Per ovviare a questo problema, la soluzione più efficiente e lineare prevede l'introduzione di una struttura dati per la memorizzazione degli stati visitati, definita a livello di codice come una lista denominata `visited`. All'avvio di ogni passo ricorsivo, la funzione registra l'URL della pagina corrente all'interno di tale struttura, trasmettendola poi come parametro alle successive chiamate. Prima di procedere alla fase di scelte dei collegamenti (passo 5), l'algoritmo esegue un'operazione di sottrazione insiemistica, rimuovendo dall'elenco dei *link* candidati tutti gli URL già presenti in `visited`.

Caching semantico A differenza delle tecniche di memoizzazione tradizionali (come il *caching* basato su chiavi e valori esatti o *hash* di stringhe), il *caching* semantico valuta il significato logico e contestuale delle richieste inviate dagli utenti. Questa progettazione risponde all'esigenza di ottimizzare il sistema evitando di rielaborare richieste rindondanti e, di conseguenza, consumare inutilmente *token* del LLM.

Il sottosistema si appoggia perciò sull'estensione *pgvector* di PostgreSQL e opera secondo una logica sequenziale:

1. **Generazione dell'embedding**: il messaggio dell'utente viene convertito in un vettore chiamato *embedding* ad alte dimensioni, capace di catturarne i tratti semantici;
2. **Ricerca di similarità**: il vettore generato viene utilizzato per interrogare la tabella di *cache* sul database relazionale. Attraverso la funzione di calcolo di similarità del coseno, il sistema confronta l'*input* corrente con le interrogazioni precedentemente storicizzate nelle ultime ore;
3. **Valutazione della soglia e raffinamento:** vengono estratti dal database i record che presentano la massima similarità riscontrata. Successivamente, per garantire la massima accuratezza ed evitare falsi positivi, viene selezionato il record semanticamente più pertinente tramite una chiamata di validazione al LLM. In base all'esito di questo controllo si determinano due scenari:
    - **Cache Hit**: il modello conferma l'equivalenza semantica; il sistema estrae la risposta pre-calcolata e la restituisce, ottimizzando tempi e costi;
    - **Cache Miss**: non viene riscontrata alcuna corrispondenza; viene avviata la normale elaborazione della richiesta.

Tale strategia di memoizzazione trasforma l'infrastruttura da un sistema puramente reattivo e computazionalmente oneroso ad uno in grado di incrementare l'efficienza e la qualità del prodotto software.

Invio delle notifiche sui passaggi di crawling Un'altra difficoltà affrontata è legata alla gestione degli eventi da inviare al *frontend* riguardanti i passaggi logici del *crawler*. Poiché l'esplorazione ricorsiva e l'elaborazione dei contenuti richiedono tempistiche computazionali non immediate, l'utilizzo di una classica architettura HTTP di tipo sincrono (*request-response*) avrebbe esposto il sistema a frequenti fenomeni di *timeout*, oltre a lasciare l'utente in uno stato di attesa

privo di *feedback* visivo. La soluzione ha dunque previsto l'implementazione di un modello di comunicazione asincrono basato su SSE. Il MCP Server Service espone un *endpoint* per la definizione di un canale di streaming con il *frontend*, consentendo di notificare in tempo reale ogni singolo stato di avanzamento della ricerca ("*Prelevata pagina X*", "*Generazione sintesi*", "*Scelta dei link*").

Search Service

Il servizio Search Service è dedicato alla ricezione e all'elaborazione dei messaggi dell'utente espressi in linguaggio naturale. Analizzando il contesto della richiesta, il servizio determina quale tool sia necessario attivare, ne estrae i parametri e invoca l'*endpoint* corrispondente sul microservizio MCP Server Service. Nonostante i requisiti del progetto richiedessero esclusivamente lo sviluppo del tool di *crawling*, il componente è stato nativamente concepito per poter gestire ulteriori tool in futuro; l'esistenza del *search service* come orchestratore e selettore dinamico delle azioni risponde proprio a questo fondamentale vincolo di estensibilità.

Iniezione della cronologia e riordino dei messaggi Una delle principali sfide riguarda la corretta ricostruzione della conversazione: per consentire all'LLM di comprendere appieno le richieste dell'utente, il Search Service non elabora il singolo messaggio isolato, bensì recupera l'intero set di messaggi presenti nella *chat*. Questa sequenza di messaggi viene strutturata e ordinata in senso cronologico decrescente (dal più recente al più vecchio): tale gerarchia informativa è fondamentale per specificare al modello quale sia il messaggio principale e l'obiettivo dell'utente, evitando che il contesto pregresso comprometta la coerenza della risposta.

Estrazione dei parametri e non-determinismo Un'altra difficoltà riguarda il non determinismo dei modelli LLM. Quando l'utente formula le richieste in linguaggio naturale, il *search service* deve garantire che l'LLM non solo selezioni lo strumento corretto, ma estragga i parametri di input (come URL, parole chiave o vincoli di ricerca) in un formato rigidamente tipizzato e privo di ambiguità.

Per risolvere tale criticità, la progettazione ha previsto l'adozione di schemi di validazione dei dati basati sulla libreria *Pydantic*. Forzando il modello a rispondere secondo uno schema JSON rigido, si è ridotto quasi allo zero il rischio di ricevere risposte malformate in caso di "allucinazioni".

Auth Service

L'Auth Service gestisce l'intero ciclo di vita dell'autenticazione e della sicurezza degli accessi. Ha la responsabilità di creare, validare e revocare i JWT (gestendo sia gli *access token* che i *refresh token*). Per la firma dei *token* si è scelto di adottare l'algoritmo asimmetrico *RS256* (*RSA* utilizzando *SHA-256*): questo approccio garantisce che solo l'*auth-service*, custode della chiave privata, possa generare e firmare *token* validi, consentendo al contempo agli altri microservizi di verificarne l'autenticità in totale autonomia tramite la rispettiva chiave pubblica, senza dover effettuare chiamate inter-servizio sincrone per ogni richiesta. A tale scopo, il servizio espone un *endpoint* chiamato "`.well-known/jwks.json`" che permette di distribuire la chiave pubblica tra i vari microservizi del*backend*.

User Service

User Service è il servizio dedicato alla creazione e alla gestione degli utenti della piattaforma. Offre le seguenti funzionalità principali:

- **Creazione utente**: crea un nuovo utente dato username, email e password. Solo gli amministratori possono creare un utente;
- **Modifica dati utente**: un utente può modificare username o password accedendo al proprio profilo;
- **Modifica permessi ai tool di un utente**: un admin può modificare i vari permessi di accesso, utilizzo della *cache* e visualizzazione dei passaggi di ricerca di un utente base;
- **Eliminazione utente**: un admin può eliminare un utente con qualsiasi ruolo;

History Service

History Service si occupa della persistenza e della sincronizzazione dello storico delle conversazioni, memorizzando le conversazioni tra l'utente e l'LLM per consentire il recupero e la consultazione delle *chat* passate. Oltre a garantire la continuità dell'esperienza utente tra i diversi accessi, questo componente svolge un ruolo chiave nel fornire al modello il contesto storico necessario per comprendere la richiesta dell'utente all'interno della stessa *chat*.

Efficienza della persistenza e disaccoppiamento guidato dal client Un problema riguarda l'impatto prestazionale derivante dalle frequenti operazioni di scrittura sul database relazionale. Per evitare che il salvataggio della cronologia generi latenza o blocchi il flusso principale di generazione della risposta, l'architettura sfrutta un disaccoppiamento basato sull'orchestrazione da parte del *frontend*. Il client, infatti, invoca inizialmente il Search Service per avviare il processo di ricerca e, solo al suo termine, effettua una chiamata verso l'*endpoint* dell'*History Service* per salvare i nuovi messaggi. Questo approccio garantisce che i tempi di scrittura sul database non interferiscano mai con la generazione della risposta.

4.2.2 Progettazione frontend

Il *frontend* rappresenta il livello di presentazione dell'applicazione, ovvero l'interfaccia diretta tra l'utente finale e i microservizi del *backend*. La reattività e la fluidità dell'interfaccia grafica costituiscono requisiti non funzionali fondamentali per garantire un'esperienza d'uso equiparabile a quella dei moderni sistemi di *chatbot*.

Poiché la pagina di *chat* deve mostrare i passaggi logici di ricerca in tempo reale (come il recupero delle pagine, la sintesi del contenuto della pagina e il filtraggio dei collegamenti) senza comportare il ricaricamento di tutta la pagina, si è scelto di adottare il pattern architetturale MVVM (Model-View-ViewModel). Questa separazione delle responsabilità permette di disaccoppiare rigidamente l'interfaccia utente (*View*) dalla logica di business e dallo stato dei dati (*Model*). Il *ViewModel* agisce come intermediario: quando un determinato oggetto cambia valore, aggiorna automaticamente lo stato della vista, garantendo una resa grafica fluida.

Controllo degli accessi e gestione delle rotte Al fine di garantire la sicurezza dell'applicazione e una corretta separazione dei permessi a livello di interfaccia utente, la logica di navigazione del *frontend* implementa un sistema di protezione delle rotte basato su componenti *wrapper* (componenti di ordine superiore) sfruttando la libreria *React Router*. L'accesso alle diverse macro-aree dell'applicazione (autenticazione, chat e pannello di amministrazione) viene regolato controllando lo stato del profilo utente mediante chiamate asincrone all'*endpoint* di sessione (`getCurrentUser`). Tale strategia garantisce che un utente del sistema possa visualizzare e interagire solamente con le viste a cui ha accesso. I moduli di controllo sono `Public Route`, `ProtectedRoute` e `AdminRoute`.

Il componente `Public Route` gestisce l'unica pagina accessibile da utenti non autenticati, ovvero la pagina di *login*. Il problema risolto da questo modulo risiede nell'impedire a un utente già autenticato di accedere nuovamente a moduli di login superflui. Qualora la verifica restituisca uno stato `'authenticated'`, il componente esegue un reindirizzamento forzato verso pagina principale di *chat*.

Al contrario, se un utente non autorizzato tenta di accedere ad una pagina diversa da quella di *login*, viene automaticamente reindirizzato a quest'ultima. L'utilizzo della proprietà `replace` è fondamentale in termini di gestione della cronologia del browser (*history stack*), poiché sostituisce l'indirizzo corrente impedendo all'utente di tornare indietro alla pagina non consentita tramite i comandi nativi del browser.

Per l'accesso alle principali funzionalità come *chat*, gestione utenti e profilo utente, sono stati sviluppati i moduli `ProtectedRoute` e `AdminRoute`. Il primo si assicura che il *token* di sessione sia valido: in caso di errore o di sessione scaduta (`'expired'`), l'utente viene reindirizzato alla pagina di *login*. Il componente `AdminRoute` estende questa logica introducendo un meccanismo di segregazione dei ruoli: dopo aver accertato l'autenticazione, il componente verifica che il ruolo dell'utente sia "admin". Se l'utente non possiede i privilegi di amministratore, avviene un reindirizzamento verso la pagina della *chat*.

Questa gerarchia strutturata garantisce che ogni tipologia di utente si trovi all'interno del proprio perimetro operativo.

4.3 Design Pattern utilizzati

L'architettura del sistema è stata strutturata adottando molteplici *design pattern* consolidati dall'ingegneria del software. L'impiego di questi modelli architetturali risponde alla necessità di garantire il disaccoppiamento tra i componenti e l'estensibilità del software.

4.3.1 Pattern Singleton

Il pattern *Singleton* è stato adottato per la gestione della classe `JwtValidation`, responsabile della validazione dei token JWT all'interno del *backend*. L'obiettivo era garantire che esistesse un'unica istanza di tale classe per tutto il ciclo di vita dell'applicazione, in modo da condividere la *cache* della chiave pubblica (`_jwks_cache`) tra tutte le richieste ed evitare chiamate ridondanti all'*endpoint* `/.well-known/jwks.json` esposto.

Il modulo `JwtValidation` definisce la propria logica di validazione; la sua istanza, invece, viene gestita tramite una metaclasse dedicata, `SingletonMeta`, in maniera tale da rendere il comportamento di *singleton* riutilizzabile e senza duplicare la logica di sincronizzazione in ciascuna di esse.

La metaclasse `SingletonMeta` mantiene un dizionario `_instances`, che associa ciascuna classe che la utilizza alla propria istanza unica, e un oggetto `_lock` per garantire la sicurezza in ambiente *multi-thread* durante la creazione. Il metodo `__call__` viene invocato automaticamente da Python ogni volta che la classe `JwtValidation` viene istanziata: se per quella classe non esiste ancora un'istanza in `_instances`, ne viene creata una sotto lock, altrimenti viene restituita quella già presente.

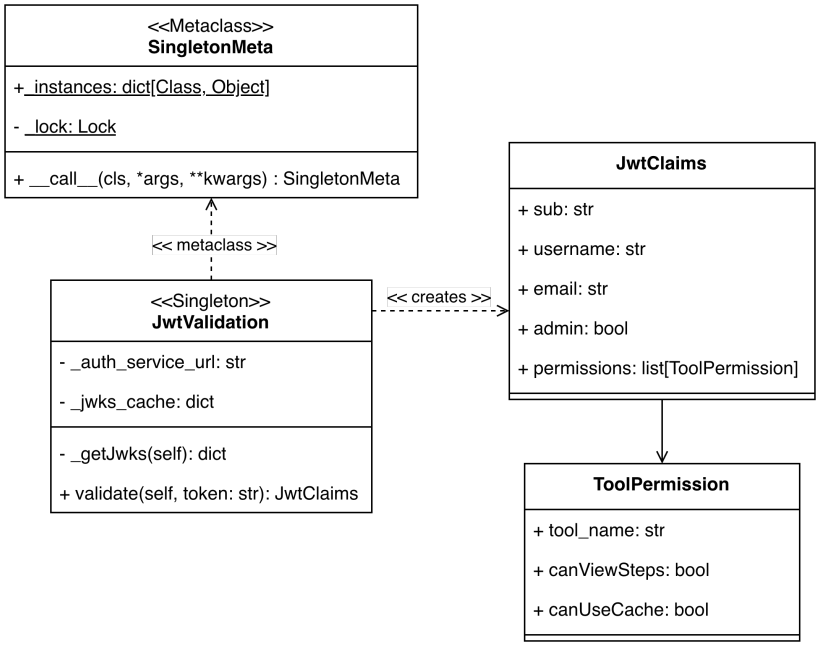

Figura 4.1: Diagramma UML del pattern Singleton implementato per `JwtValidation`

Grazie a questa architettura strutturata, il componente `JwtValidation` garantisce un unico punto di accesso globale, memorizza rigidamente in *cache* la chiave pubblica dopo il primo recupero e restituisce direttamente l'oggetto validato `JwtClaims` (correlato con il rispettivo set di `ToolPermission`), minimizzando l'*overhead* computazionale e i tempi di risposta complessivi dell'infrastruttura.

4.3.2 Pattern Adapter

Il pattern *Adapter* costituisce il meccanismo di implementazione dell'architettura esagonale (*ports and adapters*) adottata in modo sistematico in tutti i microservizi del *backend*. Ogni *port* definito nello strato applicativo (ad esempio `CachePort`, `SummarizePort`) descrive un'interfaccia astratta orientata al dominio, priva di qualunque riferimento a tecnologie o librerie specifiche. Il relativo *adapter* fornisce invece l'implementazione concreta di tale interfaccia, traducendo le chiamate provenienti dai casi d'uso in operazioni effettive verso sistemi esterni (database, chiamate al LLM, librerie). Questa separazione permette di sostituire o modificare l'infrastruttura sottostante senza impattare la logica applicativa.

A titolo esemplificativo si riportano due *adapter* implementati nel microservizio MCP Server Service: `CacheAdapter` e `SummarizeAdapter`.

La classe `CacheAdapter` implementa `CachePort` e centralizza tutte le operazioni di *caching* del microservizio. L'*adapter* in questione realizza il processo di *semantic caching* descritta nel capitolo precedente: il metodo `findCachedRequest` genera l'*embedding* della domanda tramite `OpenAIRepository`, recupera i candidati semanticamente più simili dal repository tramite ricerca per similarità vettoriale e infine invoca il metodo privato `_getSimilarQuestionIndex`, che sottopone i candidati a un modello linguistico con un *prompt* di sistema volutamente restrittivo: la corrispondenza viene accettata solo se il contenuto candidato copre *esattamente* tutte le informazioni richieste, senza inferenze o approssimazioni. Solo in caso di esito positivo viene restituito un `ToolResult` con la risposta in cache; in caso contrario il metodo restituisce `None`, segnalando un *cache miss* e lasciando che il caso d'uso proceda con l'elaborazione della risposta. La stessa logica di ricerca semantica viene utilizzata da `findCachedSummary` per la cache dei riassunti delle pagine.

La classe `SummarizeAdapter` implementa `SummarizePort` e adatta il bisogno applicativo di sintesi del contenuto verso il modello linguistico esposto da `OpenAIRepository`. Il metodo `summarize` riceve il testo estratto da una pagina web e la domanda dell'utente, costruisce un *prompt* di sistema che impone regole stringenti e restituisce il risultato, oppure `None` se il contenuto non è rilevante.

Entrambi gli adapter sono implementati come *dataclass* immutabili (`@dataclass(frozen=True)`), con le dipendenze necessarie (`CacheRepository`, `OpenAIRepository`) iniettate direttamente come campi della classe. Questa scelta, ricorrente in tutti gli adapter del sistema, mantiene gli adapter privi di stato mutabile interno, semplificandone test e sostituibilità.

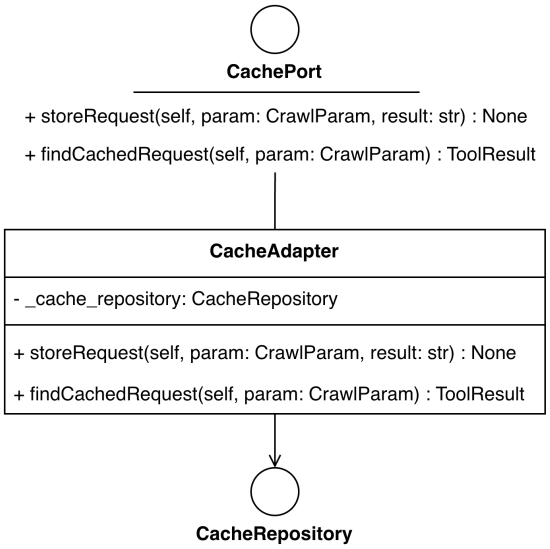

Figura 4.2: Diagramma UML del pattern Adapter implementato per `CacheAdapter`

Capitolo 5

Conclusioni

5.1 Prodotto allo stato attuale

Al termine del tirocinio è stato consegnato all'azienda un prodotto software completo e pienamente funzionante, in linea con gli obiettivi prefissati.

Il sistema, tanto dal punto di vista funzionale quanto da quello architetturale, è stato valutato positivamente dai *tutor* aziendali.

Nelle sezioni seguenti vengono descritti nel dettaglio i risultati raggiunti, illustrando le principali schermate dell'applicazione web realizzata.

5.1.1 Login

La schermata di *login* è l'unica pagina accessibile agli utenti non autenticati. Dopo l'inserimento delle credenziali (*username* e *password*), l'utente viene reindirizzato alla schermata principale, ovvero alla pagina di *chat*. In caso di errore durante l'autenticazione, viene mostrato un messaggio informativo all'utente.

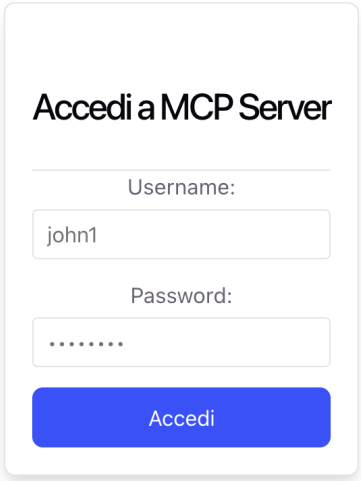

Figura 5.1: Schermata di login

5.1.2 Menu

Dopo l'autenticazione, l'utente viene reindirizzato alla pagina principale. Ogni pagina è caratterizzata da un menu laterale che consente la navigazione tra le diverse sezioni e la visualizzazione dello storico conversazionale. La scelta di integrare lo storico direttamente nel menu non è casuale: poiché la creazione e la visualizzazione delle conversazioni rappresentano il principale caso d'uso del sistema, una pagina dedicata comporterebbe un aumento delle interazioni necessarie per un'azione molto frequente. Al contrario, inserendo la lista delle *chat* nella barra laterale, queste rimangono sempre immediatamente accessibili all'utente.

Dal menu è anche possibile eseguire il *logout* dal proprio profilo tramite il bottone "Esci" posto in basso.

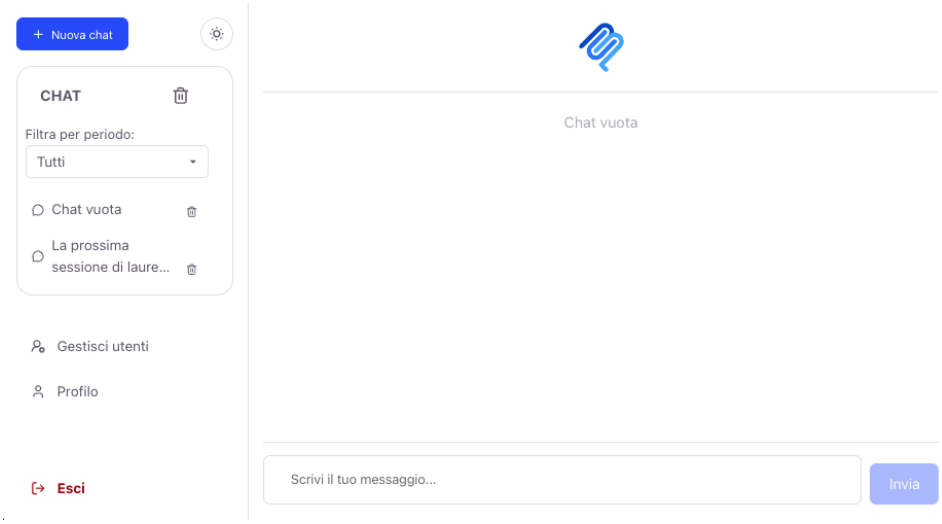

Figura 5.2: Schermata principale con menu laterale

5.1.3 Chat

Ogni conversazione è composta dallo storico dei messaggi scambiati, se presenti. Un messaggio è composto da:

- **Ruolo**: indica l'attore che ha inviato il messaggio, distinguendo tra l'utente ("TU") e il sistema ("AI");
- **Contenuto**: il testo del messaggio;
- **Data**: visualizza la data e l'orario di invio del messaggio;
- **Passaggi**: se il messaggio proviene dal sistema e l'utente possiede i permessi di visualizzazione, vengono mostrati i vari *step* di elaborazione del tool;
- **Azioni sul messaggio**: se il messaggio rappresenta una risposta generata dal sistema, sono disponibili due azioni: il pulsante di copia negli appunti e il pulsante di *download* per salvare il contenuto nel dispositivo locale. L'estensione del file scaricato è determinata dal sistema in base al tipo di contenuto: *plain text* = `.txt`, JSON = `.json`, Markdown = `.md`, HTML = `.html`.

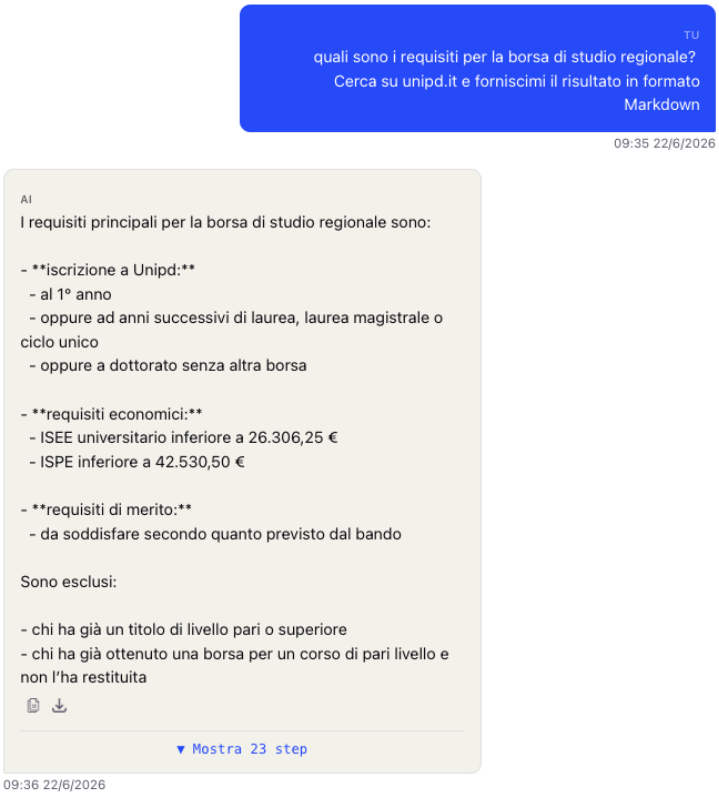

Figura 5.3: Esempio di messaggi nella schermata *chat*

Visualizzazione dei passaggi

Nell'elaborazione della risposta da parte del sistema, è possibile visualizzare tutti i passaggi avvenuti durante l'elaborazione. Questa funzionalità è fondamentale per garantire la trasparenza del processo decisionale e per mantenere l'utente informato durante l'intero processo di esecuzione del tool.
Cliccando sull'apposito elemento espandibile in corrispondenza del messaggio del sistema, l'utente può esaminare la sequenza logica delle operazioni effettuate. Per quanto riguarda il tool di web crawling sviluppato in questo progetto, i passaggi visualizzabili sono:

1. **FETCHED**: il nome del sito web da cui è stato prelevato il contenuto;
2. **SUMMARIZED**: il riassunto delle informazioni rilevanti trovate nel testo;
3. **LINKS FOUND**: l'insieme di tutti i collegamenti ipertestuali presenti nella pagina;
4. **LINKS CHOSEN**: i *link* selezionati per la continuazione del processo;
5. **ANSWERED**: comunica che la risposta è stata correttamente generata;
6. **ERROR**: visualizza l'errore avvenuto durante l'elaborazione.

Grazie a questa granularità, l'utente è in grado di verificare l'esatto percorso logico eseguito dal tool. Qualora il sistema non riuscisse a generare la risposta finale (a causa di fenomeni di allucinazione del modello o del raggiungimento dei limiti di profondità) l'utente ha comunque la possibilità di analizzare i singoli passaggi intermedi.

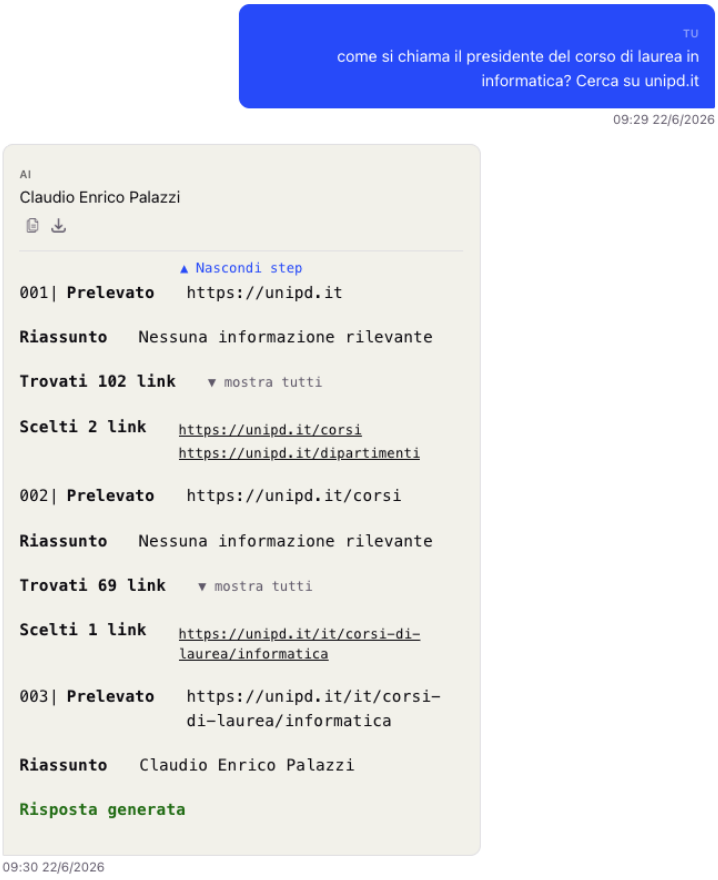

Figura 5.4: Dettaglio dell'espansione dei passaggi di elaborazione

5.1.4 Storico Chat

La sezione dello storico, integrata nella barra laterale come descritto in precedenza, consente di mantenere persistente l'esperienza d'uso dell'applicazione. Le conversazioni vengono ordinate in modo cronologico decrescente, mostrando per ciascuna il messaggio più recente della conversazione.

Attraverso questa interfaccia, l'utente può visualizzare una conversazione passata ed effettuare operazioni di gestione sulla stessa, come l'eliminazione definitiva della *chat*.

Al fine di ottimizzare la ricerca e l'organizzazione delle sessioni, l'applicazione offre inoltre una funzionalità di filtraggio temporale, che permette di isolare le conversazioni in base a tre criteri: tutte, ultima settimana e ultimo giorno. Infine, è possibile eliminare tutte le *chat* tramite il bottone dedicato, indipendentemente dal filtro in quel momento selezionato. In questo caso, l'operazione è preceduta da una finestra di dialogo che richiede all'utente di confermare l'azione, per evitare la perdita accidentale dell'intero storico dell'utente.

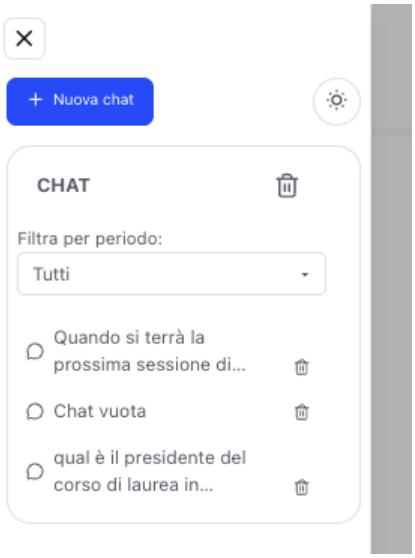

Figura 5.5: Dettaglio della gestione dello storico chat nel menu

5.1.5 Gestione utenti

La sezione di gestione utenti è accessibile esclusivamente ai profili con privilegi di amministratore (*admin*). Il pannello si presenta sotto forma di tabella riassuntiva che elenca tutti gli utenti registrati nel sistema in ordine alfabetico, mostrando informazioni quali lo *username*, il ruolo assegnato, i permessi di utilizzo dei tool e le azioni disponibili. Le regole imposte dal sistema sono le seguenti:

1. un amministratore gode di tutti i permessi di utilizzo previsti dall'applicazione e non è consentito in alcun modo revocare o limitare tali privilegi;
2. un utente base, al momento della sua creazione, non possiede alcun tipo di permesso attivo. L'*admin* ha il compito di configurare e assegnare i singoli permessi operativi in un secondo momento, abilitando l'utente alle funzionalità necessarie;
3. un amministratore possiede l'autorità di eliminare dal sistema sia gli utenti base sia gli altri profili amministrativi presenti.

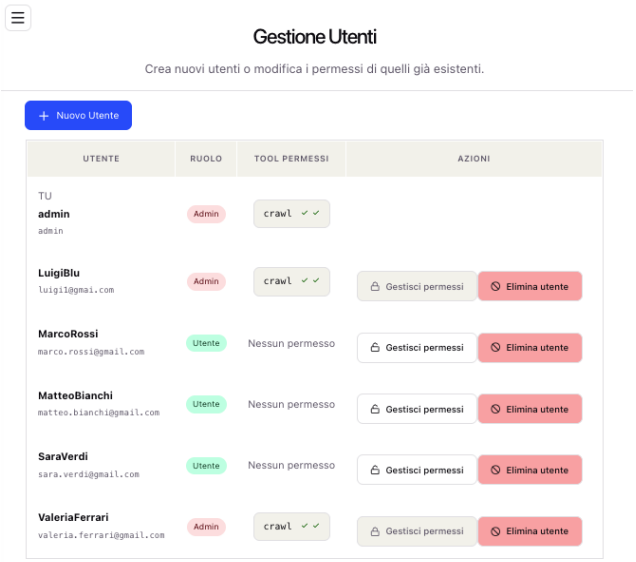

Figura 5.6: Schermata di gestione degli utenti

Creazione utente

L'amministratore ha la possibilità di creare un nuovo *account* all'interno dell'applicazione compilando un apposito modulo. Il sistema richiede l'inserimento di *username* e *email* univoci, di una *password* iniziale e del ruolo dell'utente. In conformità con la seconda regola sopra citata, il nuovo profilo verrà creato privo di autorizzazioni, costringendo l'amministratore a una successiva fase di configurazione dei permessi prima che l'utente possa interagire con il sistema (es. l'utilizzo del tool di web crawling).

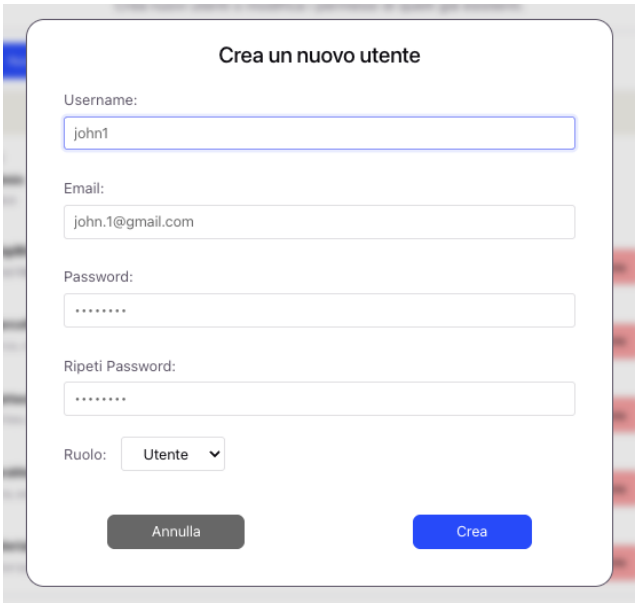

Figura 5.7: Schermata del modulo di creazione di un nuovo utente

Gestione dei permessi utente

Come descritto nel secondo vincolo, l'amministratore può modificare i permessi di utilizzo dei tool ai soli utenti base. All'interno della tabella del pannello di amministrazione è presente, in corrispondenza di ciascuna riga, un pulsante dedicato che mostra una finestra di dialogo per la gestione dei permessi. Questa interfaccia permette una configurazione granulare e mirata delle autorizzazioni. Nello specifico, i permessi che l'amministratore può abilitare o disabilitare per ciascun utente base sono:

- **[Nome Tool]**: abilita o disabilita l'accesso e l'utilizzo del tool. Nel contesto di questo progetto, avendo sviluppato solo il tool di web crawling, verrà mostrato "Crawl";
- **Utilizzo della cache**: permette al sistema di riutilizzare i dati precedentemente memorizzati in cache durante le operazioni del tool, ottimizzando i tempi di risposta e riducendo le chiamate di rete esterne;
- **Visualizzazione dei passaggi**: garantisce all'utente l'autorizzazione a espandere e consultare la sezione di dettaglio contenente gli *step* logici intermedi eseguiti dall'applicazione (es. *FETCHED*, *SUMMARIZED*, ecc.).

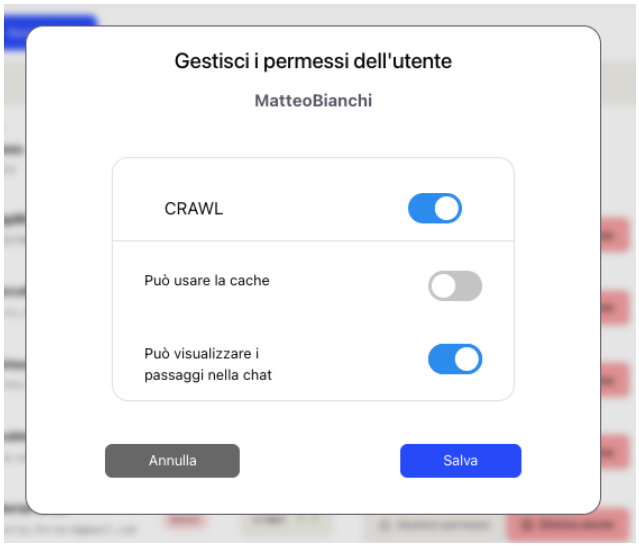

Figura 5.8: Dialogo modale per la modifica dei permessi utente

Eliminazione utente

In conformità con il terzo vincolo di sistema, l'amministratore ha la possibilità di rimuovere qualsiasi *account* (sia esso un utente base o un altro *admin*). Per prevenire perdite accidentali di dati, l'operazione non è immediata: il sistema richiede una conferma esplicita tramite una finestra di dialogo modale. Una volta confermata, l'azione comporta l'eliminazione dell'utente in maniera permanente.

Figura 5.9: Dialogo modale di conferma di eliminazione utente

5.1.6 Profilo utente

Ogni utente autenticato ha accesso a una pagina di gestione del proprio profilo personale, raggiungibile tramite il menu. In questa sezione vengono visualizzati tutti i dati dell'*account* corrente e viene offerta la possibilità di personalizzare le proprie credenziali di accesso.

Figura 5.10: Schermata di profilo

Modifica username o password

Il pannello del profilo ospita un modulo per l'aggiornamento delle credenziali, permettendo all'utente di modificare il proprio *username* o aggiornare la *password*. Per ragioni di sicurezza, il cambio della *password* è subordinato all'inserimento della *password* attuale.

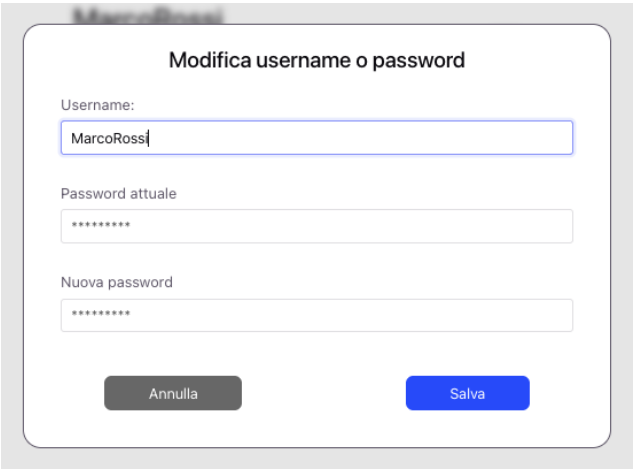

Figura 5.11: Schermata di gestione del profilo per la modifica delle credenziali

5.2 Raggiungimento degli obiettivi

In questa sezione viene discusso il raggiungimento degli obiettivi del progetto di tirocinio in base al grado di soddisfacimento dei requisiti definiti nel terzo capitolo.

5.2.1 Requisiti funzionali

Requisito	Tipo	Descrizione	Soddisfatto
RF-1	Obbligatorio	L'utente deve potersi autenticare presso il sistema	Sì
RF-2	Obbligatorio	L'utente deve inserire il proprio *username* per autenticarsi presso il sistema	Sì
RF-3	Obbligatorio	L'utente deve inserire la propria password per autenticarsi presso il sistema	Sì

Requisito	Tipo	Descrizione	Soddisfatto
RF-4	Obbligatorio	L'utente deve ricevere un errore se le credenziali inserite sono errate	Sì
RF-5	Obbligatorio	L'utente deve poter effettuare il *logout* dal sistema	Sì
RF-6	Obbligatorio	L'admin deve poter creare un nuovo utente nel sistema	Sì
RF-7	Obbligatorio	L'admin deve poter inserire una email per il nuovo utente	Sì
RF-8	Obbligatorio	L'admin deve poter inserire uno *username* per il nuovo utente	Sì
RF-9	Obbligatorio	L'admin deve poter inserire una password per il nuovo utente	Sì
RF-10	Obbligatorio	L'admin deve poter selezionare il ruolo del nuovo utente	Sì
RF-11	Obbligatorio	L'admin deve ricevere un errore se l'email inserita è già esistente nel sistema	Sì
RF-12	Obbligatorio	L'admin deve ricevere un errore se lo *username* inserito è già esistente nel sistema	Sì
RF-13	Obbligatorio	L'admin deve poter eliminare un utente dal sistema	Sì
RF-14	Obbligatorio	L'utente deve poter visualizzare il proprio profilo	Sì
RF-15	Obbligatorio	L'utente deve poter visualizzare il proprio *username* nel profilo	Sì
RF-16	Obbligatorio	L'utente deve poter visualizzare la propria email nel profilo	Sì
RF-17	Obbligatorio	L'utente deve poter modificare il proprio profilo	Sì
RF-18	Obbligatorio	L'utente deve poter modificare il proprio *username*	Sì
RF-19	Obbligatorio	L'utente deve poter modificare la propria password	Sì
RF-20	Obbligatorio	L'utente deve poter inviare un messaggio in linguaggio naturale per avviare una ricerca	Sì
RF-21	Obbligatorio	L'utente deve poter avviare una ricerca con *cache bypass*	Sì
RF-22	Obbligatorio	Il sistema deve poter estrarre dal messaggio dell'utente i parametri necessari all'esecuzione del tool richiesto	Sì
RF-23	Obbligatorio	Il sistema deve estrarre la domanda dal messaggio dell'utente per avviare la ricerca	Sì
RF-24	Obbligatorio	Il sistema deve segnalare all'utente i parametri obbligatori mancanti, impedendo l'avvio del tool	Sì
RF-25	Obbligatorio	L'utente deve ricevere un errore se l'azione richiesta non è permessa o il tool richiesto non è valido	Sì
RF-26	Obbligatorio	L'utente deve poter visualizzare la risposta generata dal sistema	Sì

Requisito	Tipo	Descrizione	Soddisfatto
RF-27	Obbligatorio	Il sistema deve notificare l'utente in caso di errore durante la ricerca	Sì
RF-28	Obbligatorio	L'admin deve poter visualizzare la lista degli utenti registrati nel sistema	Sì
RF-29	Obbligatorio	L'admin deve poter visualizzare lo username degli utenti	Sì
RF-30	Obbligatorio	L'admin deve poter visualizzare l'email degli utenti	Sì
RF-31	Obbligatorio	L'admin deve poter visualizzare il ruolo degli utenti	Sì
RF-32	Obbligatorio	L'admin deve poter visualizzare i permessi degli utenti	Sì
RF-33	Obbligatorio	L'utente deve poter creare una nuova conversazione	Sì
RF-34	Opzionale	L'utente deve poter specificare il formato della risposta (Markdown, HTML, JSON)	Sì
RF-35	Opzionale	L'utente deve poter visualizzare in tempo reale i passaggi della ricerca in corso	Sì
RF-36	Opzionale	L'utente deve poter visualizzare in tempo reale il singolo passaggio della ricerca in corso	Sì
RF-37	Opzionale	L'utente deve poter visualizzare il passaggio relativo alla pagina *web* in fase di analisi	Sì
RF-38	Opzionale	L'utente deve poter visualizzare il passaggio relativo al riassunto semantico della pagina analizzata	Sì
RF-39	Opzionale	L'utente deve poter visualizzare il passaggio relativo ai link trovati nella pagina analizzata	Sì
RF-40	Opzionale	L'utente deve poter visualizzare il passaggio relativo ai link scelti per continuare la ricerca	Sì
RF-41	Opzionale	L'utente deve poter visualizzare il nome del passaggio	Sì
RF-42	Opzionale	L'utente deve poter visualizzare la descrizione del passaggio	Sì
RF-43	Opzionale	L'utente deve poter visualizzare lo storico delle proprie *chat*	Sì
RF-44	Opzionale	L'utente deve poter filtrare lo storico per periodo temporale (ultima settimana, ultimo giorno, ...)	Sì
RF-45	Opzionale	L'utente deve poter visualizzare i singoli elementi dello storico	Sì
RF-46	Opzionale	L'utente deve poter visualizzare, per ogni elemento dello storico, l'ultimo messaggio della relativa *chat*	Sì
RF-47	Opzionale	L'utente deve poter aprire una conversazione presente nello storico	Sì

Requisito	Tipo	Descrizione	Soddisfatto
RF-48	Opzionale	L'utente deve poter eliminare lo storico delle proprie *chat*	Sì
RF-49	Opzionale	L'utente deve poter eliminare il singolo elemento dello storico	Sì
RF-50	Opzionale	L'utente deve poter eliminare tutto lo storico	Sì
RF-51	Opzionale	L'utente deve ricevere un errore se tenta di eliminare lo storico quando è vuoto	Sì
RF-52	Opzionale	L'utente deve poter visualizzare i messaggi di una conversazione	Sì
RF-53	Opzionale	L'utente deve poter visualizzare il ruolo associato a un messaggio	Sì
RF-54	Opzionale	L'utente deve poter visualizzare il contenuto di un messaggio	Sì
RF-55	Opzionale	L'utente deve poter visualizzare la data e l'ora di invio di un messaggio	Sì
RF-56	Desiderabile	L'utente deve poter avviare una ricerca con controllo della *cache*	Sì
RF-57	Desiderabile	L'utente deve poter salvare la risposta generata dal sistema	Sì
RF-58	Desiderabile	L'utente deve poter copiare la risposta generata negli appunti	Sì
RF-59	Desiderabile	L'utente deve poter salvare la risposta generata in un file locale	Sì
RF-60	Desiderabile	L'utente deve ricevere un errore se il file con la risposta generata non viene salvato correttamente	Sì

Tabella 5.1: Raggiungimento degli obiettivi: requisiti funzionali

5.2.2 Requisiti di qualità

Requisito	Tipo	Descrizione	Soddisfatto
RQ-1	Obbligatorio	Il sistema deve essere corredato da almeno un test di unità che verifichi il corretto funzionamento del *routing* dei *MCP tool*	Sì
RQ-2	Obbligatorio	Il sistema deve essere corredato da almeno un test per ogni funzionalità fornita dai *MCP tool*	Sì
RQ-3	Obbligatorio	Il codice sorgente deve essere contenuto in un *repository* versionato tramite *Git*	Sì
RQ-4	Obbligatorio	Consegnare la documentazione dei casi d'uso per l'utilizzo dell'applicativo *web*	Sì
RQ-5	Obbligatorio	Consegnare la specifica tecnica dell'applicativo *web*	Sì

Requisito	Tipo	Descrizione	Soddisfatto
RQ-6	Obbligatorio	Consegnare il manuale di utilizzo dell'applicativo *web* per gli utenti	Sì

Tabella 5.2: Raggiungimento degli obiettivi: requisiti di qualità

5.2.3 Requisiti di vincolo

Requisito	Tipo	Descrizione	Soddisfatto
RV-1	Obbligatorio	Il sistema deve essere containerizzato tramite Docker	Sì
RV-2	Obbligatorio	Il sistema deve utilizzare JWT per l'autenticazione degli utenti	Sì
RV-3	Obbligatorio	Il *frontend* deve essere sviluppato usando il framework React	Sì
RV-4	Obbligatorio	Il *backend* deve essere sviluppato usando il framework FastAPI	Sì
RV-5	Obbligatorio	Le componenti del sistema devono comunicare tramite API	Sì

Tabella 5.3: Raggiungimento degli obiettivi: requisiti di vincolo

5.2.4 Riepilogo dei requisiti soddisfatti

Come viene evidenziato dalle Tabelle 5.1, 5.2 e 5.3, tutti i requisiti obbligatori, opzionali e desiderabili sono stati soddisfatti.

In particolare, tutti i sessanta requisiti funzionali individuati nella fase di analisi (Tabella 5.1) sono stati implementati e verificati all'interno del sistema, coprendo le funzionalità di autenticazione e gestione del profilo, la gestione degli utenti da parte dell'*admin*, l'invio di messaggi in linguaggio naturale per l'avvio delle ricerche, la visualizzazione in tempo reale dei passaggi di elaborazione, la gestione dello storico delle conversazioni e il salvataggio delle risposte generate dal sistema.

Anche i requisiti di qualità (Tabella 5.2) risultano interamente soddisfatti: il sistema è coperto da test di unità e di integrazione per le principali funzionalità esposte dal MCP Server Service, il codice sorgente è versionato tramite *Git* ed è stata prodotta tutta la documentazione richiesta, comprendente i casi d'uso, la specifica tecnica e il manuale utente dell'applicativo *web*.

Infine, i requisiti di vincolo (Tabella 5.3) sono stati rispettati nella loro totalità: il sistema è completamente containerizzato tramite Docker, l'autenticazione è gestita mediante JWT, il *frontend* è stato sviluppato con il framework React e il *backend* con framework FastAPI, e la comunicazione tra le componenti del sistema avviene esclusivamente tramite API.

Il raggiungimento completo di tutti i requisiti, su tutte e tre le categorie analizzate, conferma il pieno soddisfacimento degli obiettivi definiti in fase di pianificazione del progetto.

5.2.5 Valutazione personale

L'esperienza di stage svolta presso Riskapp S.r.l. ha rappresentato per me un'occasione di crescita significativa, sia dal punto di vista tecnico che metodologico. Il progetto mi ha permesso di imparare tecnologie e approcci nuovi, di fare scelte architetturali complesse e di gestire al meglio la risorsa più importante, il tempo.

Dal punto di vista tecnico, ho avuto l'opportunità di approfondire l'architettura a microservizi, sperimentando in prima persona le difficoltà legate alla comunicazione tra servizi distinti e all'orchestrazione tramite Docker. La progettazione secondo l'architettura esagonale mi ha permesso di comprendere appieno l'importanza della separazione delle responsabilità: tale scelta si è rivelata molto utile durante le fasi di *refactoring* e di stesura dei *test*.

Un ulteriore elemento di crescita è stato il lavoro sul *frontend*, in particolare sulla gestione degli eventi SSE per la visualizzazione dei passaggi di elaborazione dell'LLM. Tale funzionalità ha richiesto di conciliare esigenze di reattività dell'interfaccia con la necessità di mantenere un'architettura ordinata, portandomi ad approfondire il pattern *MVVM*.

Nel complesso, ritengo che il tirocinio abbia raggiunto pienamente gli obiettivi formativi previsti, offrendomi un'esperienza concreta e completa dell'intero ciclo di vita di un progetto *software*, dall'analisi dei requisiti alla progettazione architetturale, fino alla codifica e al collaudo.

Sono grata per questa esperienza che l'università e l'azienda mi hanno offerto e sono sicura che le competenze apprese saranno preziose per il mio futuro professionale.

5.2.6 Possibili miglioramenti futuri

Nonostante il prodotto software consegnato soddisfi pienamente i requisiti iniziali, sono stati individuati diversi possibili miglioramenti mirati a ottimizzare le prestazioni, la scalabilità e l'esperienza utente:

- **Ottimizzazione dell'architettura a microservizi**: introduzione di un sistema di messaggistica asincrona (ad esempio tramite broker come RabbitMQ o Kafka), al fine di gestire in maniera più efficiente i picchi di carico e delle operazioni più onerose legate all'elaborazione dei modelli linguistici;
- **Supporto a nuovi formati di output**: estensione del sistema per includere la generazione di nuovi formati, come ad esempio documenti PDF, così da ampliare le possibilità di utilizzo;
- **Generazione dinamica del titolo della chat**: implementazione di un meccanismo in grado di generare automaticamente il titolo della conversazione in base al contenuto e al contesto dei messaggi scambiati, migliorando l'usabilità e l'organizzazione delle *chat*.

Appendice A

Architettura esagonale del sistema

A.1 Introduzione

L'architettura di ogni microservizio del *backend* è stata progettata seguendo il paradigma dell'architettura esagonale (*Hexagonal Architecture*, o *Ports and Adapters*). Questo approccio consente di separare nettamente la logica di dominio dalle dipendenze esterne, come database, API, framework web e servizi esterni.

L'obiettivo principale di questa scelta architetturale è garantire il rispetto dei principi SOLID e, più in generale, promuovere una progettazione modulare e manutenibile del sistema. In particolare, l'architettura adottata consente:

- **Isolamento della logica di business**, che rimane indipendente da framework, database e tecnologie esterne;
- **Rispetto del principio di Single Responsibility (SRP)**, grazie alla separazione tra dominio, applicazione e infrastruttura;
- **Rispetto del principio di Dependency Inversion (DIP)**, ottenuto tramite l'uso di *ports* e *adapters*, che permettono al dominio di dipendere da astrazioni e non da implementazioni concrete;
- **Testabilità elevata**, grazie alla possibilità di sostituire facilmente le dipendenze esterne con *mock* o implementazioni fittizie;
- **Manutenibilità ed estensibilità**, facilitando l'evoluzione del sistema senza impatti significativi sulle componenti esistenti;
- **Disaccoppiamento tra dominio e infrastruttura**, che consente di modificare dettagli tecnici (database, LLM, servizi esterni) senza alterare la logica interna al sistema.

Nel sistema sviluppato, tale paradigma è stato applicato in modo coerente a tutti i microservizi principali: *auth-service*, *user-service*, *search-service*, *history-service* e *mcpserver-service*.

A.2 Struttura generale dei microservizi

Ogni microservizio del sistema segue una struttura standardizzata, organizzata nei seguenti livelli:

- **Domain**: contiene le entità, i *value object*, i servizi di dominio e le eccezioni;
- **Application**: definisce i casi d'uso e le interfacce (ports);
- **Adapters**: implementa l'interazione con l'esterno (*inbound* e *outbound*);
- **Infrastructure**: gestisce dettagli tecnici come persistenza, DTO (*Data Transfer Object*) e integrazioni.

A.3 Architettura del progetto

La struttura del progetto segue il seguente schema ricorrente:

```
src/
  adapters
      inbound
      outbound
  application
      ports
          inbound
          outbound
      repositories
  domain
      entities
      services
      value_objects
      exceptions
  infrastructure
      persistence
      dtos
  main.py
```

Per comprendere l'efficacia del paradigma *Ports and Adapters* all'interno del sistema è necessario analizzare la responsabilità di ciascun livello e come questi interagiscono tra loro.

A.3.1 Il Core: Domain e Application

Il centro dell'esagono è costituito dal **Domain** e dal livello di **Application**. Questo nucleo non ha alcuna conoscenza del mondo esterno: non sa se i dati arrivino da una chiamata HTTP, da una riga di comando o da un test unitario. Tale livello si compone dei seguenti elementi:

- **Domain Entities e Value Objects**: rappresentano i concetti di business puri. Ad esempio, in History Service, `ConversationEntity` e `MessageEntity` contengono la struttura per la definizione della conversazione.

```python
@dataclass(frozen=True)
class ConversationEntity:
    id: UUID
    user_id: UUID
    created_at: datetime
    messages: list[MessageEntity]

@dataclass(frozen=True)
class MessageEntity:
    id: UUID
    conversation_id: UUID
    role: str
    content: str
    type: str
    format: str
    created_at: datetime
```

Listing A.1: Definizione delle classi ConversationEntity e MessageEntity

- **Domain Services**: incapsulano logiche di business complesse che coinvolgono più entità. Un esempio è il `CrawlToolService` in MCP Server Service, che coordina l'algoritmo del tool di web crawling senza legarsi a client HTTP o librerie di *scraping* specifiche.

```python
@dataclass(frozen=True)
class CrawlToolService(CrawlToolUseCase):
    cachePort: CachePort
    fetchPagePort: FetchPagePort
    summarizePort: SummarizePort
```

```
    checkAnswerabilityPort: CheckAnswerabilityPort
    chooseLinksPort: ChooseLinksPort
    formatAnswerPort: FormatAnswerPort
    notifyProgressPort: NotifyProgressPort
    manageChunksPort: ManageChunksPort

    def execute(self, param):
        # Logica di crawling

```

Listing A.2: Implementazione di CrawlToolService

- **Inbound Ports (Use Cases)**: sono interfacce che definiscono ciò che l'applicazione può fare. Nel microservizio di autenticazione, per esempio, `AuthUseCase` espone il contratto per le operazioni di login o di rinnovo del *token*.

```
class AuthUseCase(ABC):
    @abstractmethod
    def login(self, username: str, password: str) -> TokenPair :
    ...

    @abstractmethod
    def refresh(self, token: str) -> TokenPair : ...

    @abstractmethod
    def logout(self, token: str) -> None : ...

    @abstractmethod
    def getJwks(self) -> dict : ...

```

Listing A.3: Definizione di AuthUseCase

- **Outbound Ports**: sono interfacce che definiscono i bisogni infrastrutturali del dominio. Ad esempio, il dominio di MCP Server Service ha bisogno di archiviare i dati estratti o di notificare l'avanzamento dei passaggi all'utente; per farlo, definisce astrazioni come `CachePort`.

```
class CachePort(ABC):
    @abstractmethod
    def findCachedRequest(self, param: CrawlParam) -> ToolResult
    | None : ...

    @abstractmethod
    def storeRequest(self, param: CrawlParam, result: str) ->
    None : ...

```

Listing A.4: Definizione di CachePort

A.3.2 I Punti di Ingresso: Inbound Adapters

Gli *Inbound Adapters* (o *Driving Adapters*) guidano l'applicazione catturando gli stimoli esterni e traducendoli in comandi per i casi d'uso interni. Nel progetto, questi adattatori sono rappresentati dai **Controller** web FastAPI.

Prendendo come esempio la classe `AuthController` all'interno di Auth Service:

1. Il controller espone l'*endpoint* HTTP `POST /login`;
2. Riceve la richiesta formattata secondo il `LoginDto`;

3. Valida l'input sintattico e mappa il DTO in un oggetto di dominio;
4. Invoca l'*inbound port* `AuthUseCase` per eseguire la logica di autenticazione.

```python
class AuthController:
    def __init__(self, authUseCase: AuthUseCase):
        self.authUseCase = authUseCase
        self.router = APIRouter()
        self._register_routes()

    def _register_routes(self):
        @self.router.post("/login")
        def login(dto: LoginDto):
            try:
                return self.authUseCase.login(dto.username, dto.password)
            except (UnauthorizedException, UserNotFoundException):
                raise HTTPException(status_code=403, detail="Forbidden")
            except Exception as e:
                raise HTTPException(status_code=500, detail=str(e))

        [...]
```

Listing A.5: Definizione endpoint "/login" in AuthController

A.3.3 Le Connessioni Esterne: Outbound Adapters e Persistence

Gli *Outbound Adapters* (o *Driven Adapters*) implementano le interfacce definite dalle porte d'uscita, gestendo l'effettiva interazione con i dettagli tecnologici. Grazie al meccanismo della *Dependency Inversion*, le classi infrastrutturali ereditano dalle porte dell'applicazione. Un esempio lampante di questo disaccoppiamento è visibile in MCP Server Service:

- L'applicazione definisce la porta `SummarizePort` per richiedere il riassunto di un testo;
- All'esterno dell'esagono, l'adattatore `SummarizeAdapter` implementa tale interfaccia interfacciandosi con le API di OpenAI.

Lo stesso principio regola il livello di persistenza dei dati: i moduli `Repository` definiscono le interfacce e i contratti astratti di accesso ai dati, la cui implementazione concreta è demandata alle rispettive classi `RepositoryImpl` all'interno dello strato infrastrutturale. Grazie a questo totale disaccoppiamento, qualora in futuro l'azienda decidesse di sostituire l'attuale database relazionale, l'intervento di *refactoring* si limiterebbe alla sola riscrittura della classe di implementazione, lasciando il nucleo completamente inalterato.

Il medesimo vantaggio architetturale si riscontra nella gestione dei modelli di linguaggio: un'eventuale transizione verso un LLM differente da quello attuale comporterebbe esclusivamente l'aggiornamento o la sostituzione dello specifico *outbound adapter*, senza alcun impatto sulla logica di calcolo e sul comportamento globale del sistema.

Appendice B

Bibliografia

Siti web consultati

Beautiful Soup Documentation. 2026. URL: https://beautiful-soup-4.readthedocs.io/en/latest/.

Docker Compose Specification. 2026. URL: https://docs.docker.com/compose/.

FastAPI Framework Documentation. 2026. URL: https://fastapi.tiangolo.com/.

FastMCP Framework Documentation. 2026. URL: https://gofastmcp.com/.

Introducing the Model Context Protocol. URL: https://www.anthropic.com/news/model-context-protocol.

Introduction to JSON Web Tokens. 2026. URL: https://www.jwt.io/introduction#what-is-json-web-token.

LangSmith Documentation - Tracing QuickStart. 2026. URL: https://docs.langchain.com/langsmith/observability-quickstart.

OpenAI Python Library - Documentation. 2026. URL: https://pypi.org/project/openai/0.26.5/.

React Documentation. 2026. URL: https://react.dev/.

RS256 vs HS256: What's The Difference? - Learn the difference between RS256 and HS256 JWT signing algorithms. 2026. URL: https://auth0.com/blog/rs256-vs-hs256-whats-the-difference/.

Server-Sent Events - Specification for FastAPI. 2026. URL: https://fastapi.tiangolo.com/tutorial/server-sent-events/.

Vite Documentation. 2026. URL: https://vite.dev/guide/.

Acronimi e abbreviazioni

API Application Program Interface. 63

HTML HyperText Markup Language. 63
HTTP HyperText Transfer Protocol. 63

JSON JavaScript Object Notation. 63
JWT JSON Web Token. 63

LLM Large Language Model. 63

MCP Model Context Protocol. 63
MVVM Model-View-ViewModel. 63

RAG Retrieval-Augmented Generation. 63

SQL Structured Query Language. 63
SSE Server-Sent Events. 63

UML Unified Modeling Language. 63
URL Uniform Resource Locator. 63

XML eXtensible Markup Language. 63

Glossario

Caching il *caching* è una tecnica di memorizzazione temporanea di dati o risultati di operazioni già eseguite, finalizzata a ridurre i tempi di risposta e il carico computazionale evitando di ripetere elaborazioni o richieste identiche. 3, 4, 63

Docker *Docker* è una piattaforma software che consente di creare, distribuire ed eseguire applicazioni all'interno di container, ambienti isolati e leggeri che includono tutte le dipendenze necessarie, garantendo portabilità e coerenza tra diversi ambienti di esecuzione. 3, 4, 6, 8, 10, 39, 62, 63

Framework in ambito informatico un *framework* (ing. struttura, intelaiatura) è un insieme strutturato di librerie, strumenti e convenzioni che fornisce una base di partenza per lo sviluppo di applicazioni software, semplificando e standardizzando il processo di programmazione. 3, 4, 8–10, 12, 39, 42, 44, 62–64

HTML (HyperText Markup Language) *HyperText Markup Language HTML* (ing. linguaggio di marcatura per ipertesti) è il linguaggio standard utilizzato per la creazione e la strutturazione di pagine web, basato su un sistema di tag che definiscono il contenuto e la struttura semantica del documento. 2, 4, 5, 37, 43, 45, 52, 60, 63

HTTP (HyperText Transfer Protocol) *HyperText Transfer Protocol HTTP* (ing. protocollo di trasferimento di ipertesti) è il protocollo di comunicazione standard utilizzato per lo scambio di dati nel World Wide Web, basato su un modello richiesta-risposta tra client (colui che richiede i dati) e server (colui che fornisce i dati). 5, 46, 63, 65, 66

JSON (JavaScript Object Notation) *JavaScript Object Notation JSON* (ing. notazione a oggetti di JavaScript) è un formato leggero per lo scambio di dati, basato su coppie chiave-valore, ampiamente utilizzato nelle comunicazioni tra applicazioni web grazie alla sua semplicità e leggibilità. 2, 5, 11, 37, 44, 47, 52, 60, 63

JWKS (JSON Web Key Set) Formato standard per la pubblicazione di un insieme di chiavi crittografiche pubbliche, utilizzato dai servizi per recuperare le chiavi necessarie alla verifica della firma dei token JWT emessi da un'autorità di autenticazione. 44, 63

JWT (JSON Web Token) *JSON Web Token JWT* è uno standard per la creazione di token di accesso compatti e autocontenuti, utilizzati per trasmettere in modo sicuro informazioni di autenticazione e autorizzazione tra le parti coinvolte in una comunicazione. 3, 6, 12, 39, 44, 47, 49, 62, 63

LLM (Large Language Model) un *Large Language Model LLM* (ing. modello linguistico di grandi dimensioni) è un modello di intelligenza artificiale addestrato su enormi quantità di testo, capace di comprendere e generare linguaggio naturale, e alla base del funzionamento degli assistenti conversazionali moderni. 3, 6, 8, 9, 22–26, 41–43, 45–48, 50, 63, 64, 67

Markdown Linguaggio di markup leggero progettato per essere facilmente leggibile e scrivibile in formato testo semplice, convertibile nativamente in HTML o altri formati. 2, 4, 37, 52, 60, 63

MCP (Model Context Protocol) *Model Context Protocol MCP* è un protocollo standard aperto che consente l'integrazione di strumenti, fonti di dati e servizi esterni con agenti di intelligenza artificiale. Tramite *MCP* un modello linguistico può interrogare in modo standardizzato risorse esterne, come API o servizi web, ricevendo risposte strutturate e immediatamente utilizzabili. 1–7, 9–12, 41, 42, 44, 63

MVVM (Model-View-ViewModel) *Model-View-ViewModel MVVM* è un pattern architetturale per lo sviluppo di interfacce utente che separa la logica di presentazione (*ViewModel*) dalla logica di business (*Model*) e dall'interfaccia grafica (*View*), favorendo la manutenibilità e la testabilità del codice. 6, 48, 63

Prompt engineering il *prompt engineering* (ing. ingegneria dei prompt) è la disciplina che si occupa della progettazione e dell'ottimizzazione delle istruzioni testuali fornite a un modello di intelligenza artificiale generativa, al fine di ottenere risposte più accurate, pertinenti e coerenti con l'obiettivo desiderato. 3, 63

RAG (Retrieval-Augmented Generation) Tecnica di ottimizzazione dell'output dei modelli linguistici di grandi dimensioni. Tale processo estende le capacità degli LLM fornendo loro delle basi di conoscenza autorevole da parte di domini specifici, come archivi o documenti esterni. Il RAG migliora l'output del LLM in modo che rimanga pertinente, accurato e utile in vari contesti. 63

SQL (Structured Query Language) Linguaggio standardizzato utilizzato per la definizione, l'interrogazione e la manipolazione dei dati all'interno di basi di dati relazionali. 43, 63

SSE (Server-Sent Events) *Server-Sent Events SSE* è una tecnologia che consente a un server di inviare aggiornamenti in tempo reale a un client tramite una connessione HTTP persistente, permettendo una comunicazione unidirezionale continua dal server verso il client. 5, 6, 12, 42, 44, 47, 63

Tool in ambito di intelligenza artificiale un *tool* (ing. strumento) è una funzionalità esposta da un sistema esterno che un agente AI può invocare per svolgere un compito specifico, come il recupero di dati o l'esecuzione di un'operazione, ricevendone poi il risultato per proseguire il proprio ragionamento. 2, 3, 5, 6, 11, 14, 22–25, 37, 41, 43, 44, 47, 52, 53, 55, 56, 59, 63, 65

API (Application Program Interface) in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 3, 6, 12, 39, 42, 43, 62–64, 67

UML (Unified Modeling Language) in ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 63

URL (Uniform Resource Locator) Stringa testuale che identifica univocamente la posizione di una risorsa su Internet, specificandone il protocollo di accesso, l'indirizzo del server e il percorso della risorsa stessa. 46, 47, 63

Web crawling il *web crawling* (ing. scansione del web) è il processo automatizzato di navigazione ed estrazione di contenuti da pagine web, tipicamente effettuato da un software detto *crawler*, utilizzato per raccogliere e strutturare informazioni provenienti da fonti online. 2, 5, 7, 10, 14, 25, 41, 43, 53, 56, 63, 65

WebSocket il protocollo *WebSocket* consente di stabilire un canale di comunicazione bidirezionale e persistente tra client e server su una singola connessione, permettendo lo scambio di dati in tempo reale senza la necessità di ripetute richieste HTTP. 5, 63

XML (eXtensible Markup Language) Linguaggio di markup estensibile utilizzato per la rappresentazione e lo scambio di dati strutturati in un formato testuale leggibile sia da macchine che da esseri umani, organizzato secondo una struttura ad albero di elementi annidati. 43, 63