ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dipartimento di Informatica - Scienza e Ingegneria
SCUOLA DI SCIENZE

Corso di Laurea in Scienze e Tecnologie Informatiche

Sviluppo di un servizio di gestione documentale mediante elaborazione del linguaggio naturale

Relatore:
Prof. Ivan Lanese

Presentata da:
Alex Rossi
0001089916

II Sessione Dicembre 2025

Anno Accademico 2024/2025

««Un sogno non è ciò che vedi nel sonno, è ciò che ti impedisce di dormire.»» (A.P.J. Abdul Kalam)

A chi continua a inseguire i propri sogni.

Abstract

Negli ultimi anni, l'evoluzione dei modelli di intelligenza artificiale di tipo *Large Language Model (LLM)* ha aperto nuove prospettive per l'automazione dei processi aziendali e la gestione intelligente delle informazioni. Questa tesi presenta lo sviluppo e l'implementazione di un *gestore documentale intelligente* basato sul *Model Context Protocol (MCP)*, un protocollo di interfaccia che consente l'interazione strutturata tra sistemi AI e risorse esterne. L'obiettivo del progetto, che trae origine dal lavoro svolto durante il periodo di tirocinio curricolare universitario, è l'automazione dei processi di memorizzazione, analisi e consultazione di grandi volumi di documenti eterogenei, sfruttando le capacità dei modelli di AI per estrarre, sintetizzare e restituire informazioni in linguaggio naturale. L'applicazione, mediante il *server MCP*, consente quindi di caricare documenti — ad esempio fatture, bilanci o documenti di trasporto — e successivamente interrogare il sistema (tramite una *chat-AI*) riguardo informazioni specifiche sui dati archiviati.

Keywords: Gestore documentale, Model Context Protocol, LLM, server MCP, NLP.

Indice

Elenco delle figure

Listings

Capitolo 1

Introduzione

Negli ultimi anni gli avanzamenti tecnologici nel campo dell'intelligenza artificiale, e la diffusione dell'utilizzo su larga scala di strumenti di *Large Language Model*, hanno permesso la creazione di un numero sempre maggiore di servizi volti ad automatizzare compiti ripetitivi, facendo di queste tecnologie il fulcro dei programmi sviluppati.

In questa tesi viene presentata l'implementazione di un progetto che cerca di automatizzare uno dei compiti che risultano tra i più complicati per la mente umana: la memorizzazione e l'elaborazione di grandi quantità di dati. Nello specifico si presenta l'implementazione di un *gestore documentale* che pone l'uso dei servizi offerti dagli strumenti di AI, ed i protocolli associati, come fulcro del progetto stesso.

Nello specifico, il progetto presentato si basa sull'utilizzo di un protocollo sviluppato negli ultimi anni come risultato della ricerca di nuove metodologie di applicazione dell'intelligenza artificiale: il *Model Context Protocol* [24].

Come verrà espresso meglio nel capitolo 2, il Model Context Protocol è un protocollo d'interfaccia ideato per avere un'interazione fluida tra AI e risorse esterne, spostando l'attenzione dalla capacità di reperibilità del messaggio alla sua correttezza.

Spiegazione generale ed esempio d'uso

Il programma presentato in questa tesi è pensato per salvare una grande quantità di documenti (caricati direttamente dall'utente) ed offrire assistenza per quanto concerne le informazioni contenute. I file caricati possono spaziare senza alcuna restrizione su molteplici argomenti, nonostante ciò, allo stato di sviluppo attuale, l'utilizzo del programma è particolarmente indicato per documenti che trattano argomenti finanziari o aziendali (e.g. *fatture, documenti di trasporto o bilanci aziendali*).

Un **esempio d'uso** tipico è il seguente: si ponga che un'azienda di logistica voglia semplificare il processo di gestione dei dati riguardanti i propri *documenti di trasporto (DDT)*. Attraverso l'utilizzo del programma presentato in questo progetto di tesi è possibile quindi prima salvare grandi quantità di documenti (non limitandosi ai solo file documentali o tabulari, ma anche file mail o immagini in cui sono presenti dati relativi agli argomenti precedentemente citati), e successivamente, attraverso la funzionalità di *chat-AI* offerta dal programma, è possibile richiedere in linguaggio naturale informazioni presenti sui documenti precedentemente salvati (ponendo domande come *'Quale è la media dei costi totali dei documenti che ti ho caricato nell'ultimo mese?'*); oppure richiedere direttamente di scaricare sul proprio sistema specifici file (scrivendo ad esempio *'Ritornami tutti i file pdf che ti ho caricato.'*).

Per maggiori informazioni sull'uso del programma si consiglia di fare riferimento a quanto scritto nel capitolo 3, *Guida all'uso del programma*, di questa tesi.

Lavoro pregresso

L'idea del progetto presentato in questa tesi prende origine dal lavoro realizzato durante il periodo di tirocinio curricolare universitario, nel quale sono state sviluppate parti di codice che hanno posto le basi concettuali per la creazione del programma per come viene presentato nello stato attuale.

L'obiettivo originario del tirocinio era lo sviluppo di un servizio in grado di classificare automaticamente i documenti caricati, sfruttando i servizi di intelligenza artificiale offerti da Claude AI.

Nel progetto di tesi vengono impiegati i servizi di Claude che necessitano dell'*encoding* del contenuto dei file per l'analisi. Di conseguenza, il codice del progetto integra le parti relative all'encoding già sviluppate durante il tirocinio e adattate ai requisiti specifici di questa tesi.

Nello specifico, facendo riferimento alla struttura di progetto presente nella figura 4.1, nella cartella di progetto */analyzers* sono presenti le parti di codice (in linguaggio Python) sviluppate durante il tirocinio ed in seguito incluse nel progetto finale.

Per maggiori informazioni sul funzionamento del programma, consultare il capitolo 4.

Struttura della tesi

Di seguito viene presentata la struttura generale della tesi. L'obiettivo è fornire al lettore una panoramica complessiva del lavoro svolto, guidandolo nella comprensione degli argomenti trattati: in primo luogo vengono illustrate le conoscenze di base necessarie, successivamente

viene esaminata l'implementazione del progetto ed infine, dopo aver discusso i limiti attualmente presenti nell'implementazione del programma, vengono suggeriti possibili sviluppi futuri del lavoro.

Dopo questo capitolo introduttivo, sono presenti:

- **Capitolo 2**, *Nozioni preliminari*: il capitolo introduce le informazioni necessarie per la comprensione di questa tesi.
- **Capitolo 3**, *Guida all'uso del programma*: il capitolo fornisce una guida completa al *setup*, inizializzazione ed uso del programma.
- **Capitolo 4**, *Implementazione progetto*: Il capitolo illustra l'implementazione del progetto.
- **Capitolo 5**, *Conclusioni*: nel capitolo conclusivo si analizzano i limiti attualmente presenti nel programma, si presentano lavori simili e si propongono future estensioni del lavoro svolto.

Capitolo 2

Nozioni Preliminari

Nel seguente capitolo verranno introdotte tutte le informazioni necessarie affinché vengano poste le basi per la comprensione di questa tesi.

2.1 Model Context Protocol

Il *Model Context Protocol (MCP)* è il fulcro del progetto proposto, quindi è consono iniziare l'enunciazione delle nozioni tecniche partendo da quest'ultimo: così facendo si crea un filo logico che permetta al lettore di comprendere meglio l'insieme di argomenti trattati.

Come viene descritto da Hou e colleghi in [24], MCP è un'interfaccia ideata per un'interazione fluida tra modelli di *AI (Intelligenza Artificiale)* e risorse esterne.

La particolarità del MCP, come evidenzia Patil in [32], è la capacità di colmare il principale gap presente nei protocolli comuni come *TCP/IP*: la presenza di un *contesto*, ovvero la tracciabilità del flusso delle informazioni e delle operazioni eseguite (ad esempio tramite apposite annotazioni), permette l'elaborazione dei dati in modo efficiente e la riduzione di errori causati dall'ambiguità della richiesta. Ciò sposta l'attenzione sulla correttezza della richiesta e non sulla necessità di recapitare la risposta. Generalmente il *contesto* si può ricavare in molteplici modi, nel caso di questo progetto di tesi è ricavato attraverso la scrittura su file in *locale*.

2.1.1 Architettura MCP

L'interfaccia MCP si sviluppa attraverso l'implementazione di tre componenti: ***MCP Host***, ***MCP Client***, ***MCP Server***. Grazie alla figura 2.1 è possibile avere un'intuitiva rappresentazione grafica per comprendere meglio il modello descritto.

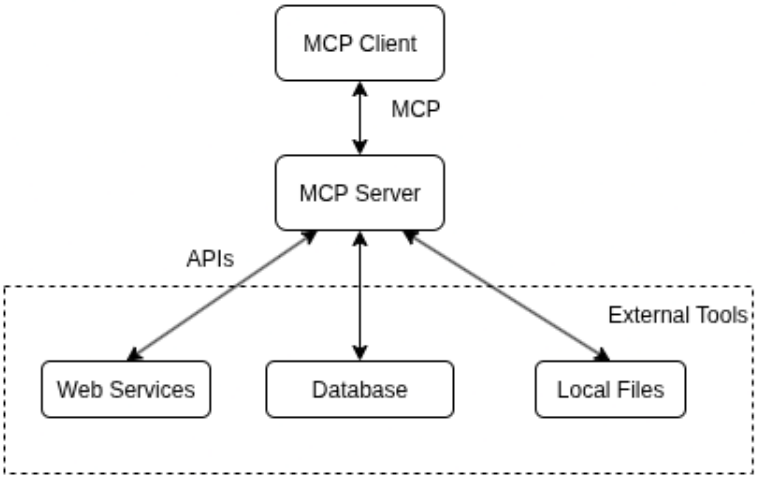

Figura 2.1: Rappresentazione grafica dell'architettura *MCP* (adattata da fig.1 in [24]).

2.1.2 MCP Host

Il *MCP Host* è l'applicazione che fornisce l'ambiente per eseguire compiti che coinvolgono l'uso sia di strumenti di intelligenza artificiale che servizi esterni. Un esempio ne sono *IDEs* (Ambienti per lo sviluppo di codice) integrati con AI oppure i recenti strumenti di *Large Language Models (LLMs)* integrati nella console del terminale.

2.1.3 MCP Client

Il *MCP Client* funge da intermediario tra MCP Host ed il MCP Server. Permette all'utente che usufruisce del servizio di visualizzare il risultato delle operazioni in tempo reale.

2.1.4 MCP Server

Il *MCP Server* è il sistema centrale ed è responsabile di: gestire la comunicazioni con **strumenti esterni**, fornire il giusto contesto ai **modelli di intelligenza artificiale** e garantire i **prompt**[1] corretti ai servizi di *AI*.

Il principale flusso di lavoro, e ciclo di vita, dei server MCP è:

1. **Fase di Creazione:** composta a sua volta da tre sotto fasi cruciali:

    - **Registrazione del Server:** viene assegnato al server un nome unico in modo tale da essere riconosciuto dal *Client* e dai componenti associati.

[1] Template predefiniti che permettono di orientare l'AI nell'esecuzione di compiti specifici sui dati forniti.

- **Installazione:** vengono installate tutte le componentistiche necessarie affinché il server lavori correttamente.
- **Verifica Integrità:** vengono controllate tutte le informazioni necessarie affinché il server possa usufruire degli strumenti esterni. Ad esempio se si usufruisce di un servizio di AI esterno, si verifica la connessione usando la *passkey* (metodologia per autenticazione dell'identità adeguata).

2. **Fase Operativa:** il server risulta operativo e pronto a soddisfare le richieste che vengono effettuate. Sono quindi richiamati sia strumenti esterni che il servizio di AI, ciò permette di creare un ecosistema *sandbox*[2] e garantire lo scambio e l'elaborazione dei dati in modo sicuro.

3. **Fase di Aggiornamento:** questa fase permette al server di tenere aggiornato il *contesto* delle informazioni che vengono trattate dall'utente attraverso tre attività chiave:
   - **Gestione delle Autorizzazioni:** verifica che l'accesso ai dati rimanga valido anche dopo una modifica.
   - **Controllo Versione:** permette di mantenere coerenza tra diverse versioni dei dati.
   - **Gestione Vecchie Versioni:** elimina dati deprecati oppure obsoleti.

2.2 Transformer

I *Transformer* sono un'architettura di rete neurale, introdotti per la prima volta nel 2017 in [50] da Vaswani e colleghi, che plasmano la base di tutti gli odierni *Large Language Models*.

2.2.1 Architettura Transformer

Come viene citato in [50] l'architettura si basa sul meccanismo di *attention*: non si usano tecniche di apprendimento automatico per sequenze, come *Reti Neurali Convoluzionali (CNN)* o *Reti Neurali Ricorsive (RNN)*, bensì sono presenti componenti che interagiscono tra loro.

I componenti principali sono quindi:

- **Meccanismo di Self-Attention:** permette al modello di riconoscere la rilevanza dell'input mediante l'attribuzione di un *peso*.

[2]Tipologia di ambiente basato sull'isolamento che garantisce sicurezza attraverso la supervisione delle operazioni effettuate.

- **Meccanismo di Multi-Head Attention:** esegue il meccanismo precedentemente citato su diverse relazioni in modo parallelo, permettendo una comprensione maggiore del problema al modello stesso.
- **Maccanismo di Positional Encoding:** a causa dell'assenza di ricorsione, si necessita di questo meccanismo per fornire informazioni sulla posizione adeguata degli elementi nella sequenza calcolata.
- **Feed-Forward Networks:** reti *completamente connesse* applicate in modo indipendente ad ogni posizione.
- **Strato di Normalizzazione** e **Connessioni Residue:** utili per stabilizzare il modello durante l'addestramento.

2.2.2 Architettura Encoder-Decoder

L'architettura del transformer si può quindi riassumere secondo lo schema *Encoder-Decoder*: L'**Encoder**, costituito da N strati di meccanismi *multi-head* secondo la struttura di rete *feed-forward*, ed il **Decoder**, con struttura analoga al precedente che, per evitare che il modello guardi erroneamente la sequenza (quindi per esempio guardi la sequenza in una posizione più avanti rispetto a quello che dovrebbe fare a tempo i), analizza l'output dell'Encoder dopo che quest'ultimo sia passato attravero strati di *cross-attention*.

2.2.3 Vantaggi dell'Architettura Transformer

I vantaggi introdotti dai *Transformer* sono svariati: **Parallelizzazione** nel processare, **Long-range Dependencies** (grazie al meccanismo di *attention* è possibile rilevare dipendenze tra i dati con un raggio molto ampio) e **Scalabilità** rispetto alla quantità di dati forniti.

2.2.4 Principali evoluzioni del concetto di Transformer

Attualmente le principali implementazioni del concetto dei Transformer si hanno con:

- **BERT (Bidirectional Encoder Representations from Transformers):** modello progettato per il pre-addestramento di rappresentazioni bidirezionali profonde da testo non etichettato. Come riportato in [16], *BERT* utilizza la parte encoder dell'architettura transformer per comprendere informazioni linguistiche semantiche e sintattiche.

- **GPT (Generative Pre-trained Transformer):** modello autoregressivo (decoder-only) usato per predire il flusso di parole future data la *sequenza passata* di parole in *dataset non-etichettati*. Dopo il pre-traning è possibile adattare il modello a compiti specifici fornendo un *dataset etichettato* ristretto.

2.3 Claude AI

Come viene descritto nella documentazione redatta dall'azienda Anthropic [3], *Claude AI*, recentemente rinominato semplicemente *Claude*, è uno dei Large Language Model più preformanti sul mercato attuale.

2.3.1 Informazioni generali su Claude

Secondo la documentazione ufficiale [4], attualmente Anthropic supporta lo sviluppo di sette modelli, ognuno con i propri punti di forza e debolezza. Generalmente però si possono ridurre i modelli a tre macro-famiglie:

- **Claude Haiku:** *Haiku* è il modello più *leggero*. Principalmente è pensato per coloro che ricercano performance su compiti specifici piuttosto che ragionamento complesso su compiti generali. Questo modello ha il costo di utilizzo minore tra quelli offerti dall'azienda.

- **Claude Sonnet:** *Sonnet* è il modello *medio*. Offre un giusto compromesso tra performance e generalità d'applicazione, oltre che al costo.

- **Claude Opus:** *Opus* è il modello più complesso e *pesante* che offre Anthropic. Ha la capacità di ragionamento maggiore tra i modelli citati, ha capacità di *Agent (modelli di AI capaci di svolgere autonomamente attività per conto dell'utente)* ed eccezionali capacità di *coding*. I costi di utilizzo sono i più alti tra i modelli offerti.

Tutti i modelli sono multilingue, hanno la *finestra di contesto* di 200.000 token (seppure attualmente sia in *beta* una finestra di contesto di 1.000.000 di token per il modello Opus), offfrono capacità di *Vision*[3] ed in base alla complessità possono offrire risposte in output di lunghezza variabile (da circa 9000 token per Haiku, a circa 64.000 token per Opus).

[3]Capacità di comprendere ed analizzare immagini.

2.3.2 Architettura di Claude

L'architettura di Claude è basata sui Transformer, nello specifico solo su Decoder, utilizzando meccanismi di self-attention per processare sequenze di *token*. Il suo funzionamento è quindi riconducibile a ciò che è stato precedentemente descritto nella sezione 2.2.1.

2.3.3 Funzionalità utili allo sviluppo del progetto

Nel progetto sviluppato in questa tesi sono risultati particolarmente utili le seguenti funzionalità:

- **Supporto all'analisi dei documenti:** nello specifico, Claude supporta l'analisi diretta, o tramite encoding, dei file PDF, immagini o file di testo. Per tutte le altre tipologie di file (come fogli excel) deve esserci necessariamente un encoding in formati supportati (come indicato in [5]).
- **Conteggio dei token usati nella richiesta:** come suggerito in [6], questa funzionalità è particolarmente utile per calcolare una stima dei costi, per operazione, durante l'uso di AI nelle operazioni del server MCP.

2.4 Python

Come viene citato sulla guida ufficiale dalla Python Software Foundation (Organizzazione non-profit che detiene il copyright a partire dalla versione 2.1 del linguaggio), *Python* è un linguaggio di programmazione interpretato, interattivo e orientato agli oggetti [37].

2.4.1 Origine storica

La prima bozza del linguaggio Python fu creata da Guido van Rossum nel 1989, ma solo nel 20 novembre 1991 si decise di rilasciare la prima versione.

Secondo la documentazione ufficiale, Van Rossum aveva come obiettivo quello di creare un linguaggio di *scripting* con sintassi simile ad *ABC*, con il quale aveva esperienza, ma che possedeva la capacità di sfruttare le *system call* di *Amoeba (sistema distribuito degli anni '90 su cui Van Rossum lavorava.)*. Accortosi del potenziale, Van Rossum decise di rendere il linguaggio non specifico per sistema Amoeba, bensì di generalizzarlo, garantendo il successo del progetto.

Il nome del linguaggio deriva dalla volontà di Van Rossum di usare un appellativo corto ed unico, in questo fu ispirato dal suo gruppo comico britannico preferito: Monty Python.

2.4.2 Caratteristiche rilevanti

La particolarità di questo linguaggio è che oltre al modello object-oriented, è possibile applicare diversi paradigmi di programmazione (come la programmazione procedurale e funzionale), rendendo il linguaggio facile da usare sia per piccoli *script* che per *codebase* più estese.

Nonostante Python sia un linguaggio interpretato, prima che il codice del programma venga eseguito su *Virtual Machine (VM)*, avviene una sotto fase intermedia dove si *compila* il codice e si salva il *bytecode* nella cartella *__pycache__* (per ragioni di efficienza e performance). Nella cartella citata è quindi presente il codice da eseguire a cui si fa riferimento. Questo viene aggiornato solo se vi sono modifiche nel *codice sorgente* e viene infine eseguito tramite VM[4].

Quindi riassumendo nell'esecuzione di codice Python si hanno due fasi:

I La **compilazione** da *codice sorgente* (.py) a *bytecode* (.pyc).

II L'**esecuzione** del bytecode a opera della *Python Virtual Machine* (PVM).

Altre caratteristiche rilevanti del linguaggio sono: la *tipizzazione dinamica*, il *Duck typing*[5] e il *Garbage Collector* basato su *Reference Counting* (meccanismo automatico di gestione della memoria che, attraverso un *contatore*, identifica e libera *oggetti* non più raggiungibili dal programma, prevenendo *memory leaks*).

Inoltre, ormai da diversi anni, il linguaggio include il supporto ad una *libreria standard* nel quale sono presenti funzionalità molto importanti come interfacce del sistema operativo o protocolli internet.

2.5 Librerie rilevanti ai fini progettuali

Di seguito viene riportato un elenco di librerie usate per la realizzazione di questa tesi, affiancate da una breve descrizione:

- **anthropic**: libreria per l'accesso all'*API REST* di Anthropic per l'uso di Claude ed i servizi *built-in* offerti [2]. Un esempio di accesso all'interfaccia fornita si ha tramite il seguente codice Python:

[4]Poiché Python basa la propria esecuzione sulla VM, rende teoricamente il codice compilato portabile tra sistemi differenti.

[5]Principio basato sulla citazione *"If it walks like a duck and quacks like a duck, then it must be a duck!"* secondo cui linguaggi di programmazione come Python non tengono conto del tipo effettivo di un *oggetto* per verificarne la compatibilità. Viene invece determinato dalla presenza di metodi analoghi o proprietà ereditate, permettendo quindi di ricavare i tipi degli oggetti a *runtime*.

```python
import anthropic

client = anthropic.Anthropic()

message = client.messages.create(
    model="claude-sonnet-4-5",
    max_tokens=1000,
    messages=[
        {
            "role": "user",
            "content": "What should I eat today?"
        }
    ]
)
print(message.content)
```

Listing 2.1: Esempio d'uso dell'interfaccia API di Claude (tratto dalla documentazione ufficiale [3]).

- **base64**: modulo della libreria standard che fornisce funzionalità per la codifica di dati binari in caratteri ASCII, decodifica e viceversa [33].
- **csv**: modulo della libreria standard che permette la lettura e scrittura di dati tabulari in formato *CSV* [34].
- **datetime**: modulo della libreria standard che fornisce funzionalità di manipolazione di *data ed ora* [35].
- **python-docx**: libreria per la lettura, la creazione e l'aggiornamento di file di Microsoft Word 2007 o versioni successive (*.docx*) [9].
- **dotenv**: libreria per la lettura di file con estensione *.env* che, leggendo coppie *key=value*, permette di impostare le variabili d'ambiente [25].
- **email**: modulo della libreria standard che permette la manipolazione ed invio di email [36].
- **extract_msg**: libreria per l'estrazione di informazioni, ed *attachments*, in file Microsoft Outlook's (*.msg*) [15].
- **fastapi**: *web framework* moderno e veloce per la creazione di API in Python [48].

- **hashlib**: modulo della libreria standard che permette di usufruire di algoritmi di *hashing* come *SHA256* o *MD5* [38].
- **io**: modulo della libreria standard che permette di lavorare con diversi tipi di *I/O* come *raw*, binario o testo [39].
- **json**: modulo della libreria standard che permette di lavorare con file *JSON* (JavaScript Object Notation) [40].
- **mimetypes**: modulo della libreria standard che permette di convertire l'*URL* del file selezionato nell'associato *MIME type* [41].
- **mongoengine**: libreria che funge da *Object-Orited Mapper* per lavorare con database *MongoDB* [27].
- **os**: modulo della libreria standard che permette di usare varie funzionalità del sistema operativo [42].
- **pandas**: libreria che fornisce strutture di dati veloci e flessibili, progettate per rendere facile e intuitivo il lavoro con dati *razionali* o *labeled* [49].
- **pathlib**: modulo della libreria standard che permette di manipolare i *paths* secondo una rappresentazione *object-oriented* del *filesystem* [43].
- **pillow**: libreria che fornisce all'interprete di Python la capacità di processare le immagini [11].
- **pydantic**: libreria che permette di validare *data* tramite *typing* [12].
- **pypdf**: libreria per la manipolazione di documenti *PDF* [20].
- **shutil**: modulo della libreria standard che permette di svolgere operazioni ad alto livello, come *copy*, su singoli o collezioni di file [44].
- **sys**: modulo della libreria standard che permette l'accesso di variabili, o funzioni built-in, all'interprete [45].
- **time**: modulo della libreria standard che fornisce funzioni che tracciano il tempo nel sistema [46].
- **typing**: modulo della libreria standard che, supportando a *runtime* i suggerimenti di *typing*, facilita la realizzazione di progetti grandi e strutturati [47].

- **uvicorn**: libreria che permette di implementare un *ASGI (Asynchronous Server Gateway Interface)* web server minimale a *basso livello* [10].

2.6 Vue.js

Come scritto in [29] *Vue.js* è un *framework* per il linguaggio di programmazione *javascript* che permette la creazione di interfacce utente ed applicazioni web a pagina singola (applicazione web che interagisce con l'utente riscrivendo dinamicamente la pagina corrente invece di caricare intere nuove pagine dal server).

2.6.1 Origini

Vue.js nasce dalla volontà di Evan You di semplificare i framework javascript, quali *Angular.js e React.js*, adattando il lavoro di creazione di *User Interface (UI)* secondo le proprie preferenze.

Il progetto nasce come libreria e viene pubblicata la prima versione nel 2014. Grazie al forte apprezzamento, attualmente Vue.js è diventato un progetto stabile con il proprio team di mantenitori [21].

2.6.2 Aspetti Tecnici

Secondo la documentazione ufficiale [51], il framework presenta tre aspetti tecnici caratteristici:

i **Sistema di reattività basato sul tracciamento delle dipendenze**: il framework traccia il *riferimento* di ogni componente nel momento in cui questo viene renderizzato per la prima volta. Successivamente aggiorna il riferimento, ed il *DOM*[6], solo se viene effettuata una modifica sul componente stesso.

ii **Virtual DOM**: il Virtual DOM è una rappresentazione virtuale del DOM che viene mantenuta in memoria. Questo componente *virtuale*, sincronizzato con quello *reale*, permette di eseguire il rendering dei componenti prima di aggiornare la finestra del browser. Ciò permette di calcolare il numero minimo di componenti da aggiornare e di conseguenza anche di effettuare il numero di manipolazioni minimo nel DOM, rendendo l'applicazione web altamente reattiva e fluida.

iii **Meccanismo di rendering dichiarativo**: i template vengono renderizzati con funzioni che restituiscono piccoli Virtual DOM. L'insieme di questi rendering costituiscono poi

[6] *Document Object Model*: rappresenta la struttura secondo cui i dati sono organizzati nelle pagine web.

il DOM per intero. In questo modo, quando avviene una modifica, si agisce solo nel re-rendering del mini-Virtual DOM appropriato e non sull'intera struttura, rendendo l'applicazione web più reattiva.

2.6.3 Librerie rilevanti ai fini progettuali

Di seguito viene riportato un elenco delle librerie, comprensivo di breve spiegazione, usate per la realizzazione del progetto:

- **axios**: client HTTP, bsato sul sistema di gestione delle richieste asincrone in Javascript tramite *promise* (per Node.js e browser), permette all'applicazione web di interagire con le API REST[7] di un server [8].
- **vue**: libreria standard di Vue.js [51].

2.7 Docker

Docker è una piattaforma open-source, basata su funzionalità del kernel Linux, per la *containerizzazione* delle applicazioni che garantisce isolamento, portabilità e stabilità.

Nello specifico i *container docker* sono unità leggere ed isolate, che eseguono un'applicazione e tutte le sue *dipendenze* (come librerie, configurazioni o file di sistema). Queste unità fungono come mini-macchine virtuali ma con il vantaggio di essere portabili, consumare poco spazio nel sistema ed avviarsi in pochi secondi (ciò è determinato anche dal fatto che l'isolamento avviene al livello del *kernel* e non a livello *hardware* come per le macchine virtuali).

2.7.1 Origini

Il progetto nasce nel 2010 da Solomon Hykescoem come strumento per gestire l'infrastruttura della *Platform as a service (PaaS)* offerta da dotCloud, la sua società co-fondata [52].

Successivamente nel 2013 la società, rinominata Docker Inc, rende la tecnologia usata open-source. Tale decisione si dimostrò cruciale per il successo di Docker a livello globale.

2.7.2 Aspetti tecnici

Come accennato precedentemente, Docker utilizza tecnologie del Kernel Linux per permettere di creare ambienti isolati denominati *container*. Nello specifico usufruisce di:

[7] *Representational state transfer* è uno stile architetturale per sistemi distribuiti.

- **Namespaces**: forniscono un meccanismo per isolare le risorse di sistema, consentendo ad ogni processo di avere la propria versione del sistema [18].

- **Cgroups (Control groups)**: progettati per controllare l'uso di risorse in un processo. Nello specifico, in Docker, serve per ridurre il rischio di *noisy neighbors*, ovvero la presenza di container ad alto uso di risorse che causano il degrado delle prestazioni di altri container sullo stesso host [14].

2.8 MongoDB

MongoDB è un servizio di database *NoSQL* (ovvero è presente la persistenza dei dati garantita da modelli *non relazionali*) orientato ai documenti: la memorizzazione dei dati avviene infatti attraverso il formato *BSON* (JSON binario) invece che *tabelle relazionali*.

Poiché non vi è imposto un modello fisso è possibile avere *collections* (ovvero raccolte di dati) di documenti con *schema* (modello secondo cui i dati vengono raccolti ed organizzati) diversi. Grazie alla flessibilità del modello NoSQL, è possibile impostare campi specifici come *indici* (struttura dati realizzata per migliorare i tempi di ricerca dei dati), permettendo la creazione di *query ottimizzate* (stringhe per la ricerca di informazioni).

2.8.1 Origini

Nel 2007 viene fondata 10gen (successivamente rinominata *MongoDB, Inc.*) con l'obiettivo di creare una PaaS (Platform as a service) composta da componenti open-source.

Poiché i database offerti sul mercato non supportavano le necessità del PaaS dell'azienda, 10gen decise di realizzarne il proprio: MongoDB [13].

Nel 2009 MongoDB, il cui nome deriva da *humongous* (da intendersi come 'enorme quantità di dati'), fu rilasciato come progetto *open-source*. Col passare degli anni il progetto divenne sempre più supportato e complesso, rendendo MongoDB uno dei database No-SQL più usati globalmente.

2.8.2 Aspetti tecnici

Seguendo ciò che viene descritto nella documentazione ufficiale [31], attualmente MongoDB supporta **pluggable storage engine** consigliando come predefinito *WiredTiger*.

WiredTiger è basato sul meccanismo di **multiversion concurrency control (MVCC)**: le *read* possono operare su *snapshot* (immagini immutabili del database) coerenti, mentre gli *update* (modifica ai dati) possono provocare *write conflicts* solo se le versioni non corrispondono.

Vengono inoltre adottate meccanismi di **optimistic concurrency control** (tecnica di gestione della concorrenza basata sul presupposto di rara presenza di conflitti nelle transazioni) dopo aver effettuato update, prima di attuare la *commit* (operazione che conferma le modifiche attuate), si effettua una fase di validazione. Si verifica quindi che nel frattempo nessuno abbia alterato il dato che si è modificato (ad esempio attraverso l'uso di meccanismi di *timestamp*, usato per tracciare temporalmente le modifiche sui dati). Se non ci sono conflitti si effettua la commit, altrimenti si fa *rollback*, ovvero porta il database ad uno stato coerente, e solo dopo permette di rifare l'operazione designata. Altre caratteristiche rilevanti sono:

- **Transazioni multi-documento** che garantiscono proprietà *ACID* (Atomicità, Consistenza, Isolamento e Durabilità) delle operazioni sui sui dati.
- **Lock a livello di documento**: il documento modificato è *bloccato* fino al commit. Se un'altra sessione tenta di modificare lo stesso documento, la transazione viene abortita e ritentata.
- **Replica set**: insieme di istanze che detengono una copia sincronizzata della stessa base dati.
- **Sharding (Partizionamento orizzontale)**: permette di distribuire i dati su più server (*shard*), ciascuno dei quali è solitamente un *replica set* (copia dei dati). Questo meccanismo supporta il dimensionamento orizzontale su grandi dataset e alti carichi. Si sceglie inoltre una *shard key (campo del documento)* su cui basare la distribuzione: i documenti vengono spartiti in *chunk* (porzioni di dati basati sulla shard key) e distribuiti tra gli shard.

Capitolo 3

Guida all'uso del programma

Nel seguente capitolo viene fornita una guida completa al *setup*, inizializzazione ed uso del programma.

3.1 Panoramica del programma

In questa sezione viene presentata al lettore una panoramica generale del programma del progetto di tesi, evidenziando funzionalità e vantaggi d'uso del programma.

Descrizione del programma

Il programma presentato funge da *raccoglitore intelligente di documenti*: oltre a consentire l'aggiunta e la rimozione di file da una determinata repository, come avviene nei comuni sistemi di archiviazione, permette di ricevere assistenza nella ricerca di informazioni sui file caricati grazie all'integrazione di servizi di intelligenza artificiale. L'utente può quindi porre domande in linguaggio naturale riguardanti i file precedentemente salvati e ottenere risposte coerenti con le proprie richieste.

Il programma è stato sviluppato con l'obiettivo di facilitare e assistere l'utente nel recupero di informazioni specifiche sui file salvati, offrendo al contempo un'interfaccia minimale, semplice e intuitiva. Il principio alla base è quello di ridurre i compiti ripetitivi, come la ricerca manuale dei documenti, e automatizzare i processi tramite strumenti di AI di ultima generazione.

Le tipologie di file supportate dal programma sono molteplici: l'utente può caricare documenti (ad esempio *.pdf*), immagini (*.png*, *.jpeg*, *.gif*), file testuali (ad esempio *.txt*), file tabulari (come *.xls*, *.odt*) e messaggi di posta elettronica (*.msg*, *.eml*).

Esempio d'uso del programma

Si consideri, ad esempio, il caso in cui l'utente desideri conoscere la *media delle fatture caricate nel programma nell'ultimo mese*. Poiché tale valore non è esplicitamente indicato nei documenti interessati, l'utente dovrebbe procedere manualmente a individuare tutti i documenti caricati nel mese corrente, annotare l'importo totale riportato in ciascuno di essi e calcolare successivamente la media.

Con il programma proposto, invece, è sufficiente porre la domanda direttamente nella chat AI: il sistema elabora la richiesta, genera automaticamente la query per ricavare i documenti pertinenti, estrae i dati necessari (secondo il modello con cui i documenti vengono memorizzati nel database) e fornisce all'utente la risposta desiderata. All'occorrenza, se richiesto, rende inoltre disponibili per il download i documenti coinvolti nella ricerca.

Funzionalità del programma

Il programma permette all'utente di creare *categorie* personalizzate garantendo l'associazione di una definizione (mediante una frase di senso compiuto) all'appellativo stabilito.

L'utente può caricare documenti al fine di salvarli in database. Il programma effettua prima una fase di *processing* dei dati: tramite servizio di AI vengono estrapolate le informazioni più rilevanti dal documento e successivamente viene assegnata, dal programma, una categoria di appartenenza (tra quelle precedentemente definite dall'utente).

L'utente può usufruire di una *chat intelligente*: ponendo domande in linguaggio naturale, l'AI risponde fornendo le informazioni richieste e, se necessario, rende disponibile al download i documenti pertinenti ai quesiti posti.

Vantaggi uso del programma

Usando il programma presentato l'utente può concentrare le sue attenzioni sul formulare correttamente la domanda in linguaggio naturale da porre all'AI, invece che occuparsi dell'interpretazione diretta dei dati sui file immagazzinati.

Si prenda come esempio un programma generale di salvataggio di file: se l'utente ha caricato decine (o centinaia) di file, troverà l'azione di ritrovare uno specifico file (oppure la rielaborazione di dati presenti in molteplici file) molto complicata e dispendiosa. Col programma presentato in questo progetto di tesi il problema si evita in quanto, come già detto precedentemente, si delega il problema agli strumenti di AI che garantiscono automazione e rapidità di risposta.

3.2 Prerequisiti

Affinché il programma risulti funzionare correttamente, prima di effettuare l'installazione di quest'ultimo, è necessario verificare di possedere alcuni requisiti.

Prerequisiti per l'esecuzione in locale

Se si decide di non usare l'installazione Docker, e quindi si effettua l'esecuzione del programma in **locale**, si necessita di un sistema con:

- Versione di Python 3.12.8 o successiva (affinché il *backend funzioni*).
- Versione di Node.js 22.19.0, o successiva, e npm 10.9.2+ (necessario per il funzionamento del *client frontend*).
- Connessione a servizi di database Mongo (come il servizio di online database mongo: MongoDB Atlas [30]).
- Versione di Git 2.34.1+ (necessario per scaricare il progetto dalla repository GitHub [1]).

Prerequisiti per l'esecuzione tramite Docker

Se invece si usano esclusivamente i servizi forniti da **Docker**, si necessita semplicemente della versione di quest'ultimo pari a 28.4.0 o successive.

Prerequisiti generali

Per usufruire dei **servizi di AI**, sia che si decida di effettuare l'installazione in locale o su Docker, si necessita anche di una *API-KEY* di Claude ottenibile da [7].

3.3 Setup programma

In questo paragrafo vengono fornite le istruzioni per svolgere il corretto setup del programma, sia che si voglia fare l'esecuzione in locale o che si esegua tramite Docker.

Accertatisi di possedere i requisiti indicati in sezione 3.2, per effettuare il setup in del programma è necessario eseguire i seguenti passaggi in ambiente *Linux*:

1. **Download dei file dalla *repository* GitHub**: questo è facilmente ottenibile aprendo il terminale di sistema e digitando:

```
$ git clone https://github.com/Axelredx/Bechelor-Thesis-Project.git
$ cd Bechelor-Thesis-Project
```

Listing 3.1: Clonazione di un progetto GitHub mediante il comando `git clone`.

2. **Creazione ed inizializzazione del file *.env***: necessario per usufruire dei servizi di AI (Claude) e database (MongoDB).

```
$ cd backend
$ touch .env
$ echo 'ANTHROPIC_API_KEY=your_api_key db=your_MONGO_DB host=
    your_MONGO_HOST port=your_MONGO_PORT username=your_MONGO_USER
    password=your_MONGO_PASS' > .env
$ cd ..
```

Listing 3.2: Creazione e setup del file di *enviroment*.

Se si sta effettuando un'installazione in **locale**, in aggiunta ai passaggi sopraindicati bisogna anche effettuare:

1. **Download delle dipendenze di progetto nella cartella */backend***.

```
$ cd backend
$ python -m venv my_project
$ source my_project/bin/activate
$ pip install -r requirements.txt
$ deactivate
$ cd ..
```

Listing 3.3: Download delle dipendenze backend.

2. **Download delle dipendenze di progetto nella cartella */frontend***.

```
$ cd frontend
$ npm i
$ cd ..
```

Listing 3.4: Download delle dipendenze frontend.

3.4 Avvio programma

Eseguiti i passaggi di setup indicati nella sezione 3.3 il programma ora è pronto all'avvio. I procedimenti da eseguire differiscono in base alla propria decisione di eseguire il progetto in locale o tramite installazione Docker.

3.4.1 Avvio locale

Se si ha deciso di eseguire il progetto in locale bisogna aprire due terminali di sistema sulla directory di progetto */Bechelor-Thesis-Project*. Sul primo terminale bisogna digitare:

```
$ cd backend
$ source my_project/bin/activate
$ uvicorn main:app --reload
```

Listing 3.5: Inizializzazione locale del backend.

Grazie al processo creato da terminale, il server si avvierà su *http://localhost:8000/* (**nota**: se la *porta 8000* non risulta disponibile, chiudere il processo che ne usufruisce e ripetere il passaggio indicato).

Sul secondo terminale invece basta digitare:

```
$ cd frontend
$ npm run dev
```

Listing 3.6: Inizializzazione locale del frontend.

Ora l'applicativo risulta funzionante ed è possibile usufruirne aprendo il browser all'indirizzo *http://localhost:5173/* (la *porta* potrebbe differire da quella indicata se è già in uso dal sistema, in tal caso apparirà sul terminale l'indirizzo esatto a cui fare riferimento).

3.4.2 Avvio tramite Docker

Se invece si è optato di usufruire del programma tramite installazione Docker basta semplicemente aprire il terminale di sistema (sempre sulla directory del progetto */Bechelor-Thesis-Project*) ed eseguire i comandi:

```
$ chmod +x ./start.sh ./stop.sh ./clean.sh
$ sudo systemctl start docker # Oppure aprire Docker Desktop
$ ./start.sh
```

Listing 3.7: Inizializzazione del progetto tramite Docker.

Il processo di inizializzazione tramite Docker richiede svariati minuti poiché, oltre a creare i vari ambienti *sandbox*, deve anche scaricare le *immagini d'ambiente* (con le conseguenti dipendenze) dal sito ufficiale.

Se tutto si è svolto correttamente senza la presenza di errori ora, nel terminale, si ottene una schermata simile a quella riportata nella figura 3.1.

Figura 3.1: Stato dei container Docker in esecuzione.

Ora è possibile usufruire correttamente del programma accedendo al browser digitando l'indirizzo *http://localhost:8080/* nella barra di ricerca.

3.5 Uso del programma

Nella seguente sezione si fornisce una guida all'utente su come utilizzare correttamente il programma. Per facilitarne la comprensione si accompagnano al testo esempi ed immagini sul flusso di lavoro.

3.5.1 Primo avvio

Durante il primo avvio l'utente, accedendo al programma tramite browser come indicato nella sezione 3.4, si troverà davanti alla schermata riportata nella figura 3.2.

La pagina iniziale presenta in alto una *barra di navigazione* con le voci *'Home'*, *'Upload File'* e *'Settings'*:

- **Home** riporta alla pagina corrente. Questa è la pagina principale ed integra una chat diretta coi servizi di AI. Le richieste che possono essere effettuate possono comprendere sia la restituzione di file specifici, che la rielaborazione di informazioni di carattere generale sui file caricati dall'utente.
- **Upload File** è la pagina responsabile per il caricamento dei file dell'utente nel database.
- **Settings** è la pagina dedicata alla modifica delle impostazioni secondo indicazioni dell'utente. Qui possono essere definite, aggiunte o eliminate le *categorie* di file che devono essere riconosciute dal servizio di AI. Inoltre è possibile eliminare documenti secondo richiesta specifica dell'utente.

In basso della pagina iniziale si trova lo spazio dedicato alla chat con l'AI. Come si può notare dalla scritta *'Definisci la categoria dei documenti nella sezione /settings per abilitare*

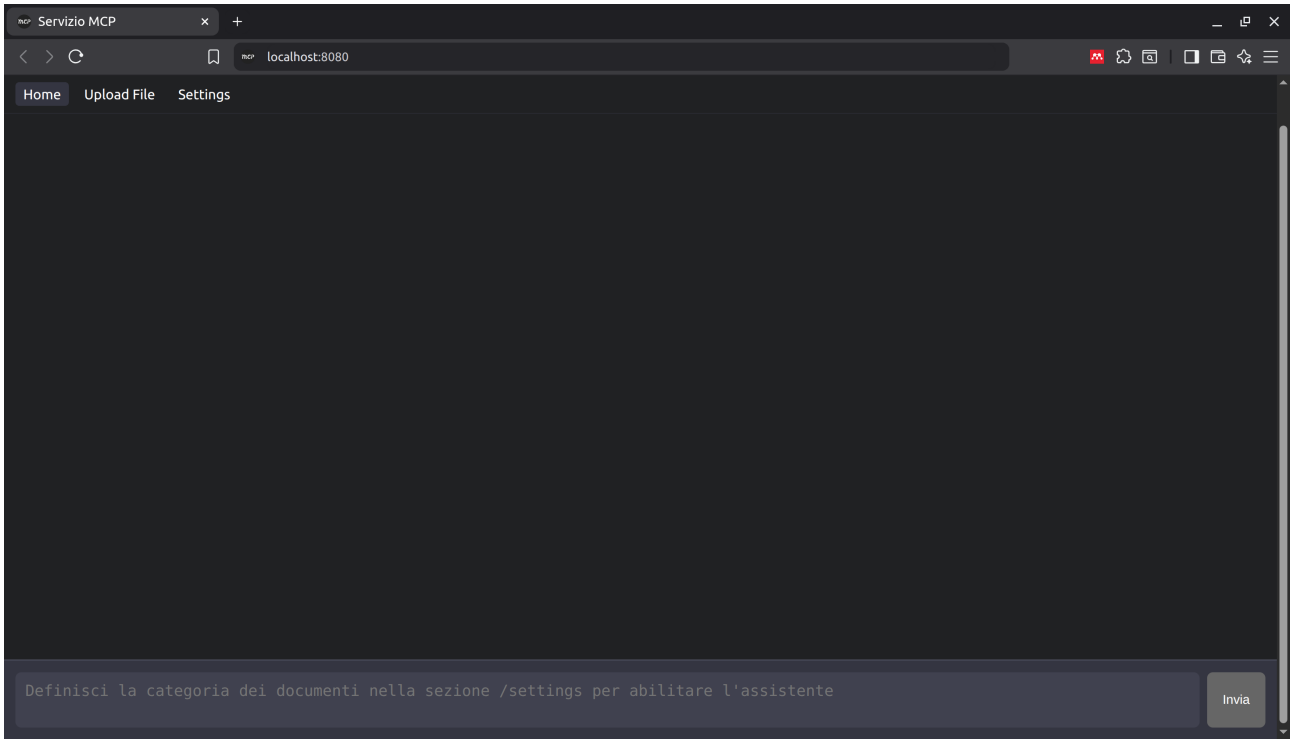

Figura 3.2: Schermata iniziale del programma al primo avvio.

l'assistente' la chat attualmente è disattivata. Infatti per evitare un uso improprio del programma da parte dell'utente, la chat rimane disattivata fino a quando si svolgono le seguenti azioni nell'ordine proposto:

1. Si **definiscono le categorie** apposite nella sezione */settings*.
2. Si effettua l'**upload di almeno un file** nella sezione */upload*.

3.5.2 Creare, aggiungere ed eliminare categorie

Per *categoria* si intende un sostantivo seguito da una descrizione di quest'ultimo. La definizione delle categorie è un'azione fondamentale: senza categorie dell'utente, il servizio di AI non riuscirebbe a classificare (ed estrarre informazioni) in modo corretto dai documenti caricati nella sezione *upload*.

Poiché il servizio è stato sviluppato con l'obiettivo di lasciare più libertà possibile all'utente, potenzialmente si possono definire delle categorie con dei sostantivi incongruenti a patto che siano accompagnati da una descrizione dettagliata (**Esempio generale**: *'AGR: tutti i documenti che riguardano fatture agricole'*).

Per definire le *categorie* bisogna quindi cliccare sulla voce '*Settings*' della barra di navigazione. Se è la prima volta che si definisce una categoria, l'utente si troverà davanti alla schermata riportata nella figura 3.3.

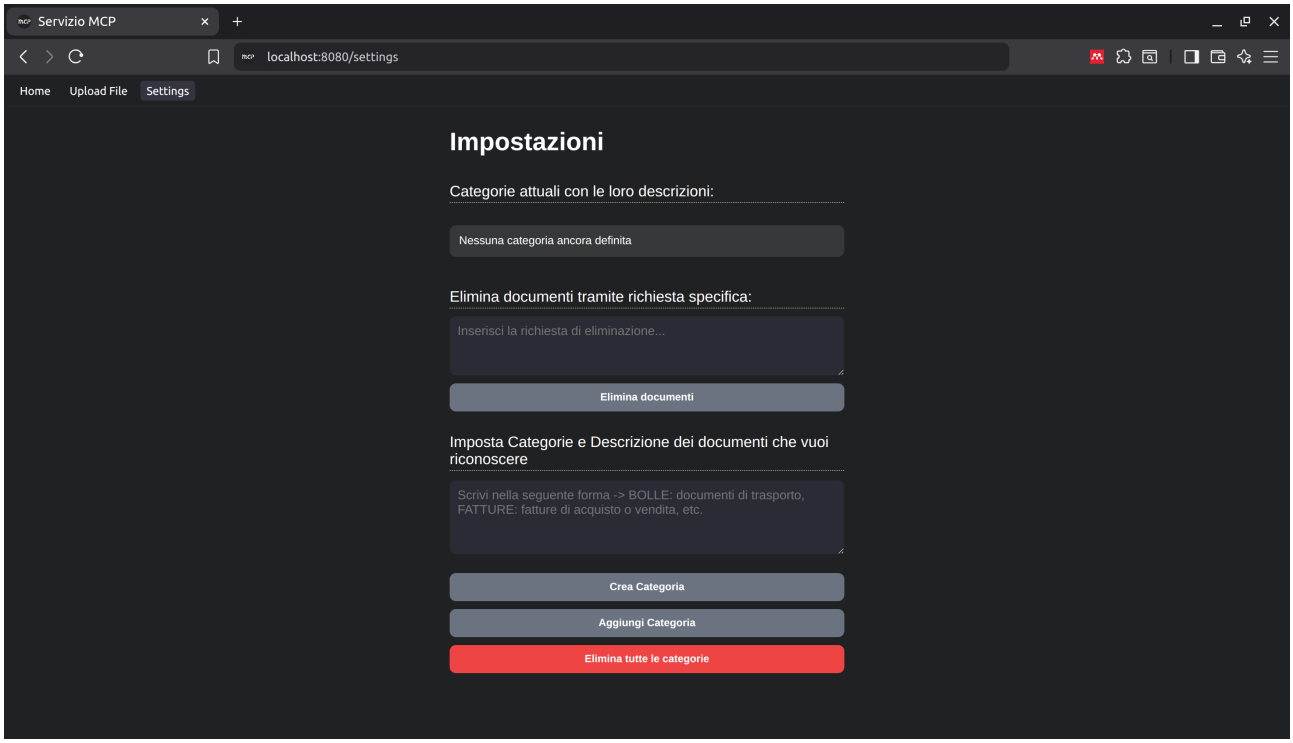

Figura 3.3: Schermata */settings* del programma.

Creare ed aggiungere categorie

Se nel database non sono presenti categorie, in alto apparirà la frase *'Nessuna categoria ancora definita'*, l'utente dovrà quindi crearne di nuove nella sezione sottostante: seguendo gli esempi che sono riportati nel template di scrittura, si definiscono quindi una o più categorie nella forma *'CATEGORIA: descrizione della categoria, ...'*.

Se è la prima volta che si definiscono categorie, cliccare sul pulsante *'Crea Categoria'*, altrimenti usare il pulsante *'Aggiungi categoria'* per aggiungere una o più categorie a quelle già presenti nel database.

Ora le categorie definite appariranno nella sezione in alto nella forma *'CATEGORIA1, CATEGORIA2, ...'*.

Eliminare categorie

Se non si è soddisfatti delle categorie definite, si può cliccare sul bottone *'Elimina tutte le categorie'*. Questa azione cancella tutte le informazioni relative alle categorie nel database riportando il programma allo stato descritto nella ad inizio di questa sottosezione.

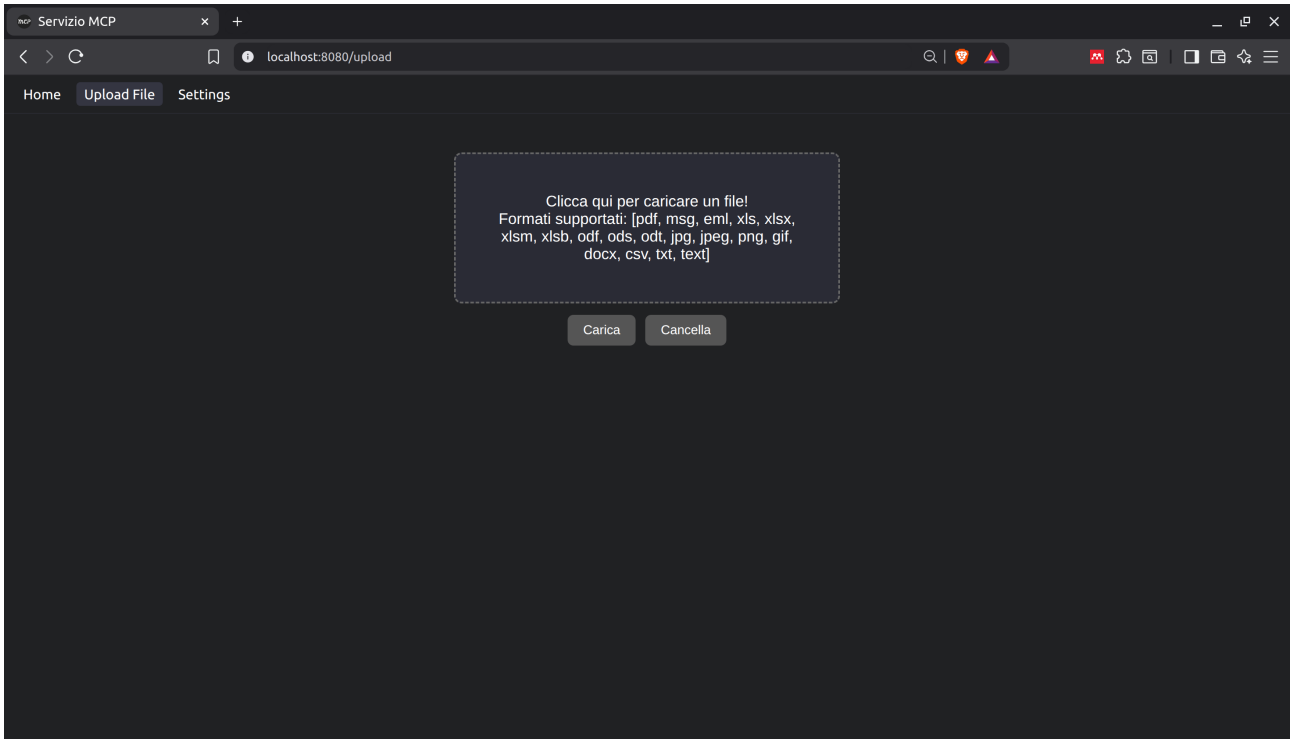

Figura 3.4: Schermata */upload* del programma.

3.5.3 Caricare documenti

Per caricare documenti bisogna andare sulla pagina */upload* cliccando sulla voce *'Upload File'* della barra di navigazione situata in alto a sinistra.

Eseguita questa azione, presupponendo che sono presenti nel database delle categorie correttamente definite, l'utente si troverà nella schermata riportata nella figura 3.4.

Per caricare un file basterà quindi cliccare nel riquadro apposito: ora è possibile scegliere il file da caricare navigando nel *filesystem* del proprio dispositivo. Successivamente basterà cliccare sul pulsante *'Carica'* per iniziare il processo di caricamento del file nel database.

Si giunge quindi allo stato indicato nella figura 3.5: è presente una *barra di caricamento* di colore azzurro. Fino a quando la barra di caricamento rimane a schermo vuol dire che il documento sta venendo processato dal servizio di AI, il quale categorizza ed estrae informazioni rilevanti dal documento, prima di essere caricato nel database.

Se tutto si è svolto correttamente senza l'insorgere di errori, dopo che il documento viene effettivamente caricato nel database, scomparirà la *barra di caricamento* e si presenterà a schermo (per qualche secondo) un messaggio che indica il successo dell'operazione (come riportato nella figura 3.6).

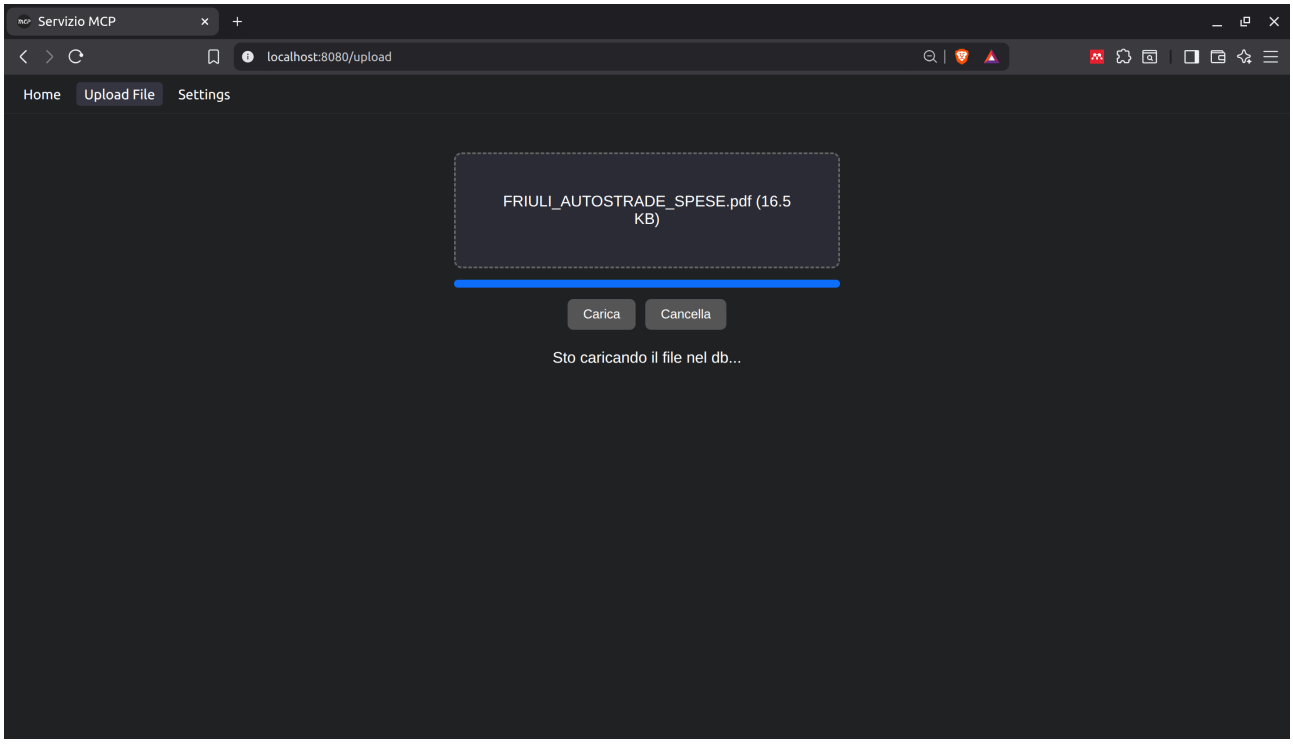

Figura 3.5: Schermata */upload* del programma: *processing* del documento.

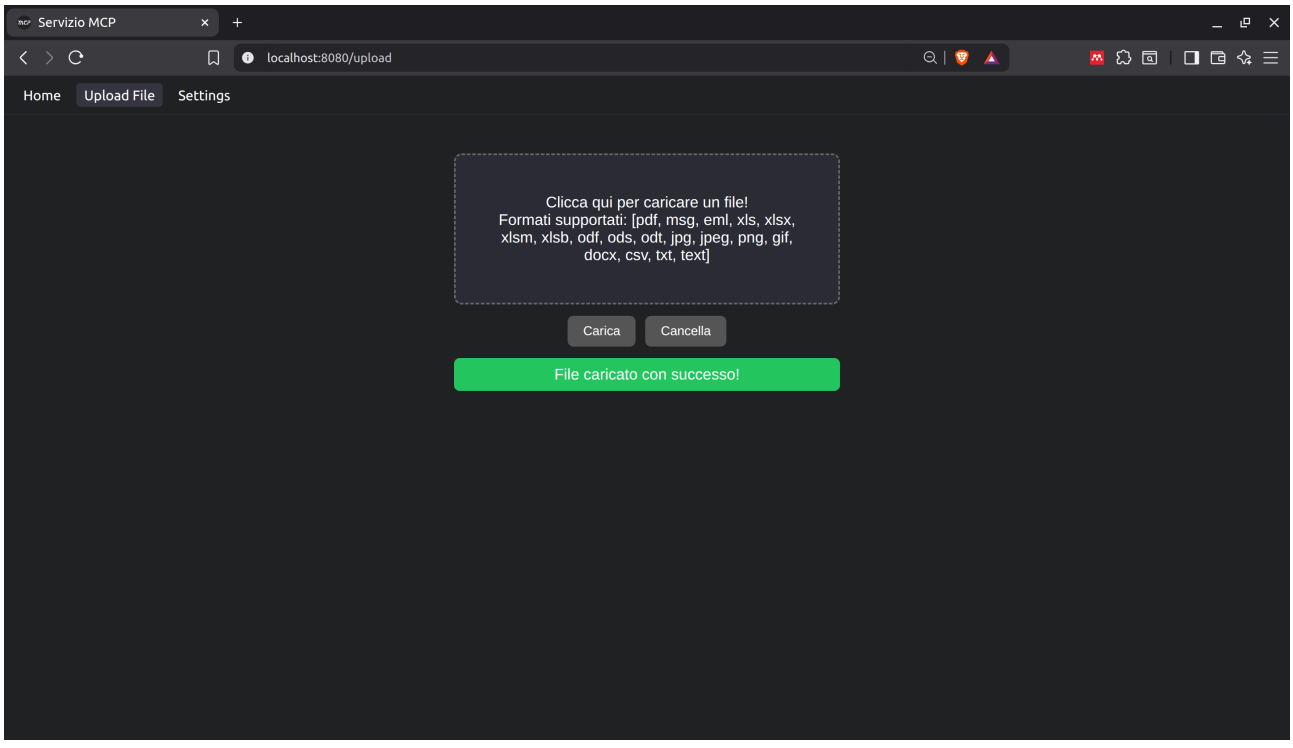

Figura 3.6: Schermata */upload* del programma: successo dell'operazione di caricamento.

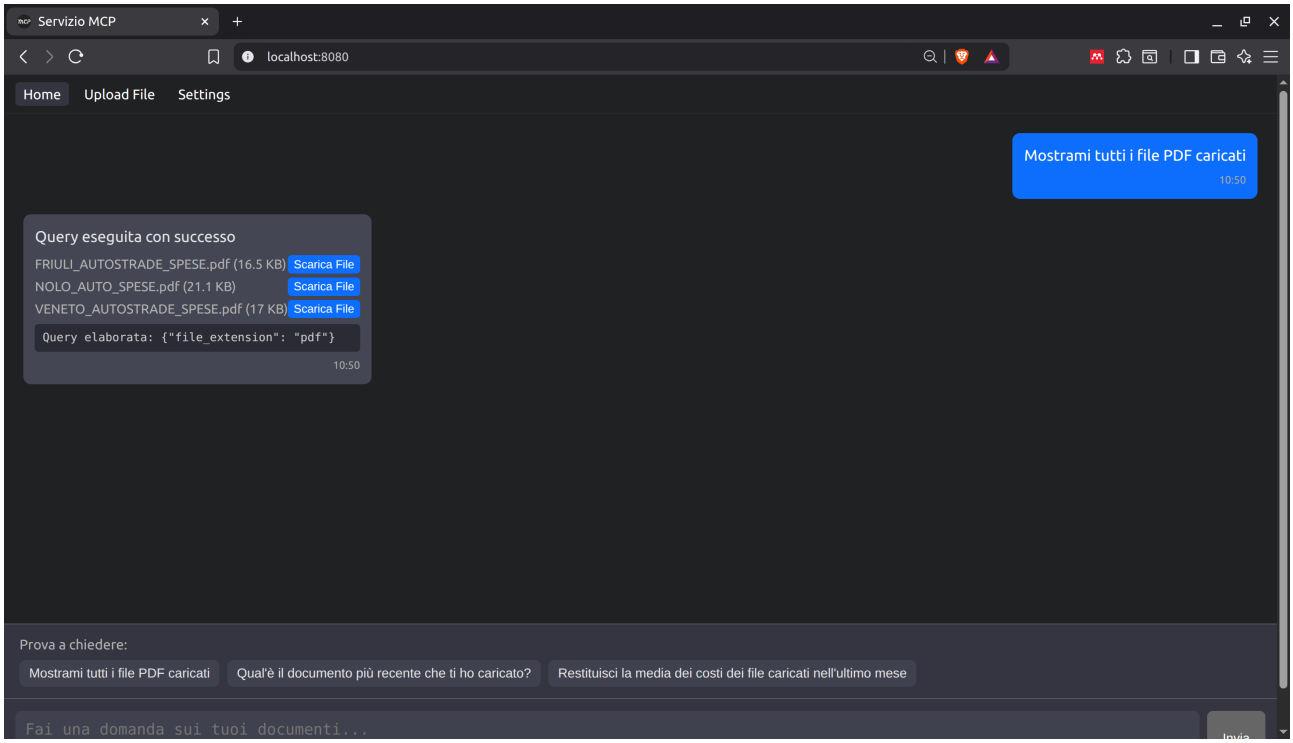

Figura 3.7: Schermata */Home* del programma: richiesta di documenti al servizio di AI.

Formati documenti supportati

Come si può notare dal riquadro nella figura 3.4, i formati dei documenti caricabili nel programma sono: *.pdf, .msg, .eml, .xls, .xlsx, .xlsm, .xlsb, .odf, .ods, .odt, .jpg, .jpeg, .png, .gif, .docx, .csv, .txt, .text.* Se l'utente carica un documento diverso da uno dei formati sopraindicati, viene inibita automaticamente la possibilità di caricare il documento nel programma ed insorge a schermo un messaggio di errore di colore rosso.

3.6 Uso del servizio di assistenza documentale AI

Dopo avere definito le categorie e caricato documenti nelle sezioni apposite, è ora possibile usufruire della funzione di *chat-AI* nella pagina principale. Alcune esempi di richieste sono ora visibili sopra la sezione dedicata alla chat.

Le richieste effettuabili dall'utente si possono dividere in due macro famiglie:

I Le **richieste di documenti**: l'utente chiede al servizio di AI di restituirgli documenti secondo quanto richiesto. Un esempio d'interazione di questo tipologia si può osservare nella figura 3.7.

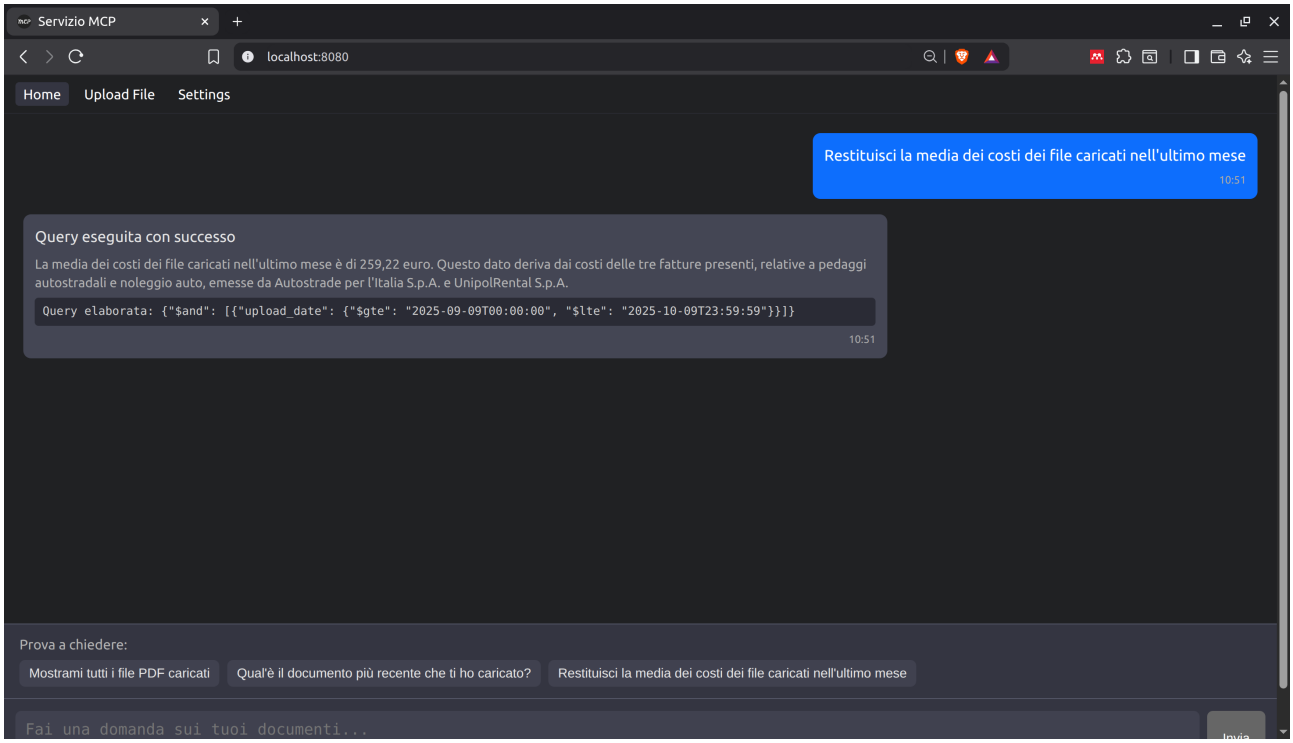

Figura 3.8: Schermata */Home* del programma: richiesta di informazioni sui documenti caricati.

II Le **richieste di informazioni sui documenti**: l'utente chiede al servizio di AI informazioni specifiche sui documenti che ha caricato. La richiesta può inoltre comprendere una rielaborazione, da parte dell'AI, delle informazioni riguardanti più documenti immagazzinati nel database. Nella figura 3.8 è riportato un esempio di questa tipologia interazione.

Forma del messaggio di risposta

Di seguito viene fornita una descrizione esaustiva del messaggio del servizio di AI come risposta della richiesta inviata dall'utente.

Facendo riferimento a quanto descritto nella sezione 3.6, i messaggi risposta possono essere:

- **Tipologia *I***: il messaggio risponde quindi alla richiesta dell'utente di tipologia *I*. Vengono forniti la lista dei file richiesti, compresi nome e dimensione del file, con la possibilità di scaricarli sul proprio sistema tramite *click* del bottone *'Scarica File'* (figura 3.7).

- **Tipologia *II***: il messaggio di risposta soddisfa le richieste dell'utente di tipologia *II*. Viene fornita una spiegazione in linguaggio naturale delle informazioni richieste dall'utente (figura 3.8).

In entrambe le tipologie di risposte sono inoltre presenti, in fondo al messaggio, le *query* (direttamente formulate dal servizio di AI) per ricavare le informazioni effettuate nel database.

Ciò risulta particolarmente utile per verificare se il servizio formula richieste sufficientemente precise da soddisfare le necessità dell'utente.

3.7 Eliminare documenti

Per eliminare documenti è necessario:

1. **Cliccare** nella sezione ***Settings*** della barra di navigazione.
2. **Scrivere in linguaggio naturale** quali documenti si vuole eliminare (anche tutti).
3. **Cliccare** il bottone '***Elimina documenti***'.

3.8 Terminare il programma

Facendo riferimento alla sezione 3.4, la metodologia per terminare il programma differiscono in base al metodo di avvio che si è deciso di usare.

3.8.1 Terminare il programma localmente

Se si ha avviato il programma localmente basterà terminare i processi creati sui rispettivi terminali:

- sul primo terminale (su cui è avviato il processo del server) con directory */Bechelor-Thesis-Project/backend*, premere contemporaneamente i tasti *CTRL* e *C*.
- sul secondo terminale (su cui è avviato il processo del client) con directory */Bechelor-Thesis-Project/frontend*, premere contemporaneamente i tasti *CTRL* e *C*.

3.8.2 Terminare le istanze di Docker

Se invece si ha avviato il progetto tramite installazione Docker basterà eseguire sul terminale di sistema, nella directory */Bechelor-Thesis-Project*, il comando:

```
$ ./stop.sh
```

Listing 3.8: Terminare le istanze Docker del programma.

Il risultato di questa operazione da output simile a quello che viene riportato nella figura 3.9.

Figura 3.9: Rimozione dei container Docker in esecuzione.

Nonostante l'operazione, è garantita la persistenza dei dati: i file salvati non vengono quindi persi alla terminazione delle istanze Docker.

3.8.3 Eliminare i container Docker

Se si vuole eliminare i container creati da Docker, ed effettuare quindi una pulizia totale, si può eseguire sul terminale anche il comando:

```
$ ./clean.sh
```

Listing 3.9: Eliminare i container Docker del programma.

Eseguire il comando sopracitato con **attenzione**: infatti così facendo verranno eliminati tutti i *container* che riguardano il progetto compresi tutti i dati salvati al loro interno (Documenti, categorie, etc.).

Capitolo 4

Implementazione progetto

In questo capitolo si discutono la struttura, le funzioni e le scelte architetturali di questo progetto di tesi. Per una migliore comprensione si accompagna il testo di esempi, riflessioni e parti di codice direttamente estratte del programma stesso.

Il codice completo (avente dimensione totale di circa 3000/3500 righe) a cui si fa riferimento in questo capitolo è disponibile sotto licenza *Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0)* su GitHub [1] e segue la struttura riportata nella figura 4.1.

```
Bechelor-Thesis-Project/
├── backend/
│   ├── ai_microservices/
│   ├── analyzers/
│   ├── controllers/
│   ├── DUMP/
│   ├── LOGFILE/
│   ├── models/
│   ├── routers/
│   ├── utility/
│   ├── main.py
│   └── ...
├── frontend/
├── doker-compose.yml
├── LICENSE
└── ...
```

Figura 4.1: Struttura semplificata della gerarchia di progetto.

4.1 Architettura progetto

Il progetto è stato sviluppato secondo la filosofia *Model-View-Controller*. Questo pattern è basato sulla separazione dei compiti fra i componenti citati:

- **Model**: fornisce i modelli secondo cui i dati vengono salvati nel database.
- **View**: visualizza i dati contenuti nel model e reagisce reattivamente alle azione eseguite dagli utenti.
- **Controller**: fornisce i metodi per permettere di modificare i dati forniti dal model.

Grazie alla filosofia *MVC*, ed il paradigma *Model Context Protocol*, il programma interagisce secondo quanto riportato nella figura 4.2.

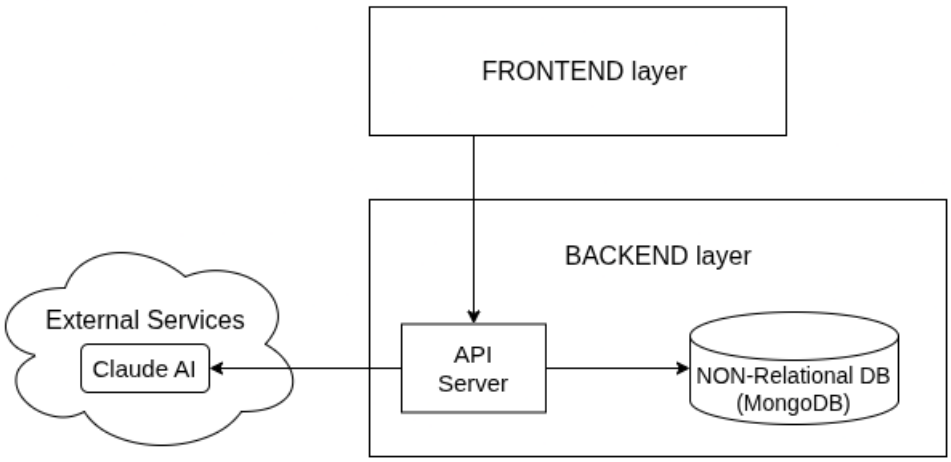

Figura 4.2: Architectural design del progetto proposto.

Le richieste effettuate dall'utente passano quindi prima dal client MCP (ovvero attraverso la *View*), che tiene traccia delle operazioni eseguite dell'utente, e successivamente vengono inoltrate al server MCP (che opera sui dati attraverso il *Controller*) per soddisfarle.

A supportare il server è inoltre presente:

- Un **servizio esterno di AI** (Claude): il servizio viene chiamato dal server per eseguire molteplici compiti di analisi e valutazione.
- Un **database NoSQL** (MongoDB): il servizio è necessario per garantire la persistenza di dati salvati dall'utente (che salva i dati seguendo i *Models* definiti).

Framework e linguaggi di sviluppo

Facendo riferimento alla struttura di progetto presentata nella figura 4.1, il *server MCP*, il cui codice di progetto è presente nella cartella */backend*, è stato sviluppato interamente in linguaggio Python. Nello specifico si è usato il framework *FastAPI* per la creazione del servizio di API.

Nella cartella */frontend* è presente la parte di progetto dedicata allo sviluppo del *client MCP* tramite *Single Page Application*. Per lo sviluppo di questa porzione di codice è stato utilizzato il framework Javascript *Vue.js*.

Paradigma di sviluppo API

Le API, che permettono di interagire con i servizi offerti dal server, sono state sviluppate seguendo lo standard REST (REpresentational State Transfer) e la filosofia CRUD: si effettua la chiamata *post* per rappresentare *create*, la chiamata *get* per rappresentare *read*, la chiamata *put* per rappresentare *update* e la chiamata *delete* per rappresentare *delete*.

L'implementazione delle API è presente nella cartella */routers* dove sono disponibili i tre file: *asker.py* (che include tutte le routes che riguardano le interazioni dell'utente con la chat-AI, oltre ad offrire controlli sui documenti presenti nel database), *classifier.py* (che include la route che gestisce il caricamento di un file nel programma) e *type_ descriptor.py* (che include tutte le routes che riguardano la manipolazione ed il salvataggio delle categorie nel database).

4.2 Flusso operazioni utente

In questa sezione viene fornita una visione generale, attraverso l'uso di diagrammi *flowchart*, delle operazioni eseguibili dall'utente riguardanti le funzionalità principali del programma. Nello specifico si analizzano le operazioni di *file upload* (figura 4.3) e di *richiesta di informazioni* (figura 4.5).

4.2.1 File upload

Nella figura 4.3 è riportato il flusso di operazioni che avvengono ogni qualvolta l'utente carica un documento per essere salvato nel database:

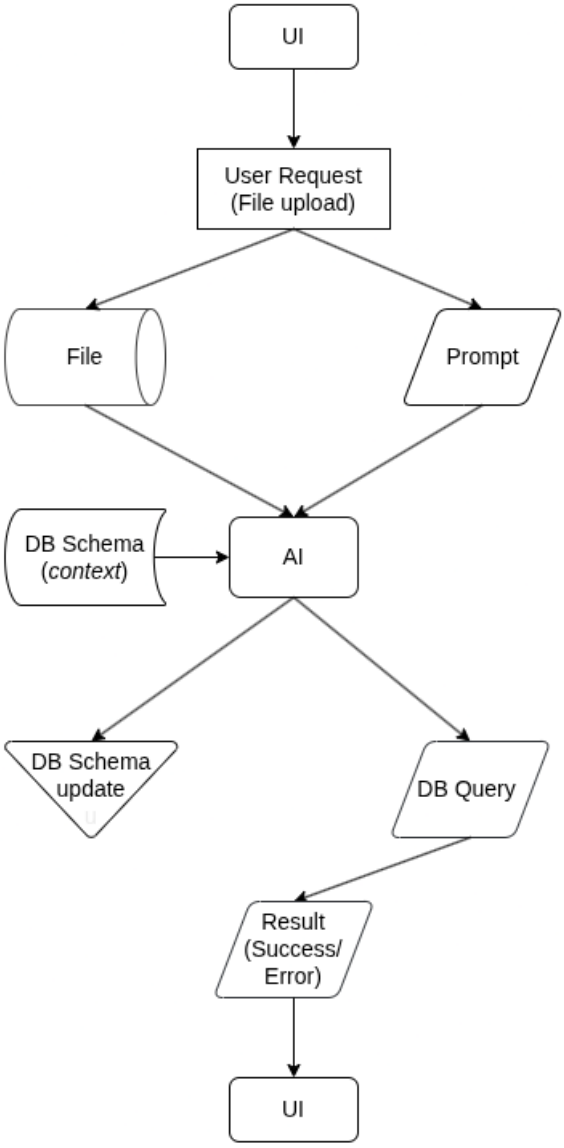

Figura 4.3: Flowchart: upload file.

I. Caricamento del documento nel programma

L'utente effettua la richiesta di **upload file tramite User Interface**. Viene quindi richiamata l'API **@router.post("/upload-file/")** per gestisce l'intero processo di caricamento del file in database. Nello specifico, tra le varie operazioni eseguite sul file prima che venga salvato nel database, si effettua l'estrazione delle informazioni rilevanti e la classificazione (per la spiegazione completa delle operazioni elencate consultare la sottosezione 4.3.2).

II. Salvataggio del file in *locale*, *encoding* ed invio a Claude

Il documento viene ora **salvato in locale su */DUMP***. Prima di inviare file a Claude, è necessario però effettuare l'encoding dei file attraverso i moduli presenti nella cartella /analyzers.

Attualmente Claude consente il caricamento diretto, tramite interfaccia e apposita elaborazione dei dati, solo di file in formato *.pdf* e di *immagini*. Negli altri casi, come verrà illustrato in seguito, è necessario adottare una soluzione alternativa per consentire l'analisi dei dati contenuti nei file: il metodo utilizzato consiste nell'effettuare l'encoding dei file in stringa, così da inserire le informazioni direttamente nel prompt passato all'AI.

Nello specifico le estensioni dei file supportate per l'encoding sono: **.csv**, **.docx**, **.eml**, **.gif**, **.jpeg**, **.jpg**, **.msg**, **.odf**, **.odt**, **.pdf**, **.png**, **.text**, **.txt**, **.xls**, **.xlsb**, **.xlsm** e **.xlsx**.

Generalmente è possibile quindi riassumere l'encoding dei file in:

- **Encoding dei file pdf**: Tramite la funzione `analyze_pdf()` (in *pdf_ analyzer.py*), è possibile effettuare l'encoding in stringhe di file pdf: per questioni di performance, se il file risulta avere più di 10 pagine (attraverso la funzione `__check_pdf_pages_number()`) si crea una copia con un numero ridotto di pagine secondo i limiti indicati (attraverso `__chop_pdf_pages()` si ottiene la copia con presenti solo le prime 10 pagine, ovvero le pagine in cui si presuppone la presenza di dati rilevanti). Successivamente si applica l'encoding della copia creata (il `chopped_file`): questo sarà il file che viene inviato a Claude per l'analisi. Nonostante ciò il file che viene salvato in database rimane quello originale presente nella cartella */DUMP*.

- **Encoding dei file documentali e testuali**: Tramite funzioni apposite (presenti nei file *csv_ analyzer.py*, *docx_ analyzer.py* e *txt_ analyzer.py*) è possibile effettuare l'encoding, a stringa singola, del contenuto dei file. Per ragioni di performance, e per ragioni legate al numero massimo di token che è possibile usufruire durante una richiesta a Claude, la stringa ritornata contiene al massimo 10000 caratteri. Se sono presenti un numero maggiore di caratteri la stringa viene troncata secondo i limiti indicati.

- **Encoding delle immagini**: L'encoding delle immagini (ovvero i file con estensioni *.gif*, *.jpeg*, *.jpg* e *.png*) in stringhe, è eseguibile tramite la funzione `analyze_img()` presente nel file img_analyzer.py. Se durante il processing l'immagine risulta avere dimensione del file maggiore rispetto ai limiti consentiti da Claude (la quale dimensione massima attuale è 5mb) viene richiamata la funzione `__compress_image()` per comprimere l'immagine. La compressione eseguita (ovvero la compressione *Lossy*) diminuisce quindi la qualità dell'immagine da 1.00 a 0.85. Il processo iterativo viene ripetuto fino a quando la dimensione dell'immagine è inferiore o uguale ai limiti consentiti.

- **Encoding delle email**: Tramite funzioni apposite è possibile effettuare l'encoding in stringhe dei file in formato *.eml* e *.msg*. Nello specifico si estraggono informazioni (come mittente, destinatario, data, oggetto ed allegati presenti) prima aprendo il file in binario e poi eseguendo le operazioni di estrazione tramite libreria apposita. Inoltre, per i file in formato *.msg*, vengono salvati anche gli *attachments* tramite *metadata*.

- **Encoding dei file tabulari**: Tramite l'apposita funzione, presente in *xls_analyzer.py*, è possibile effettuare l'encoding dei file tabulari (con ad esempio estensione *.odt* o *.xls*) in stringhe. Nello specifico vengono salvati, tramite libreria apposita, i metadati e le informazioni presenti sui fogli presenti nel file caricato. Per ragioni di performance vengono considerati solo i primi 3 fogli tabulari: in questi vengono considerati solo i dati presenti nelle prime 15 colonne e le prime 30 righe. Inoltre, per permettere una migliore comprensione dei dati a Claude, per le celle con valori `NaN` (Not a Number) oppure risultanti vuote, si inserisce invece il valore `'[EMPTY]'` nella corrispondente stringa finale. Si consideri, ad esempio, un foglio tabellare utilizzato per tenere traccia delle fatture giornaliere: disponendo i *giorni sulle righe* e i *mesi sulle colonne*, è possibile inserire nelle rispettive *celle* gli importi delle fatture emesse, lasciando vuote (oppure segnando *0,00*) quelle relative ai giorni in cui non ne sono state registrate (come rappresentato nella figura 4.4). Nel listing 4.1 viene mostrato l'encoding, in formato stringa, dell'esempio illustrato in precedenza.

# Giorno	Gennaio	Febbraio	Marzo	Aprile	Maggio	Giugno	Luglio	Agosto	Settembre	Ottobre	Novembre	Dicembre	TOTALE
1	€ 250,50		€ 180,00	€ 0,00	€ 420,75	€ 0,00	€ 0,00	€ 310,20	€ 0,00	€ 275,00	€ 0,00	€ 195,50	€ 1.631,95
2		€ 315,80	€ 0,00	€ 0,00	€ 0,00	€ 285,50	€ 0,00	€ 0,00	€ 425,30	€ 0,00	€ 0,00	€ 0,00	€ 1.026,60
3	€ 175,00	€ 0,00	€ 0,00	€ 560,00	€ 0,00	€ 0,00	€ 380,50	€ 0,00	€ 0,00	€ 0,00	€ 290,75	€ 0,00	€ 1.406,25
4		€ 0,00	€ 420,30	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 215,60	€ 340,00	€ 0,00	€ 480,25	€ 1.456,15
5		€ 485,00	€ 0,00	€ 0,00	€ 325,80	€ 0,00	€ 0,00	€ 155,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 965,80
6	€ 390,75	€ 0,00	€ 0,00	€ 225,50	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 475,30	€ 0,00	€ 1.091,55
7		€ 0,00	€ 295,00	€ 0,00	€ 0,00	€ 410,25	€ 0,00	€ 0,00	€ 0,00	€ 520,00	€ 0,00	€ 0,00	€ 1.225,25
8		€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 625,50	€ 0,00	€ 380,00	€ 0,00	€ 0,00	€ 0,00	€ 1.005,50
9	€ 125,00		€ 0,00	€ 355,75	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 265,00	€ 745,75
10	€ 0,00	€ 540,00	€ 0,00	€ 0,00	€ 0,00		€ 0,00	€ 445,80	€ 0,00	€ 0,00	€ 310,50	€ 0,00	€ 1.296,30
11	€ 0,00	€ 0,00	€ 0,00		€ 490,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 490,00
12	€ 0,00		€ 375,50	€ 0,00		€ 0,00	€ 235,75	€ 0,00	€ 0,00	€ 295,00	€ 0,00	€ 0,00	€ 906,25
13	€ 455,25	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 330,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 510,00	€ 1.295,25
14	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00		€ 0,00	€ 580,50	€ 0,00	€ 0,00	€ 0,00	€ 580,50
15	€ 0,00	€ 275,50	€ 0,00	€ 615,00		€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 425,75	€ 0,00	€ 1.316,25
16	€ 0,00	€ 0,00		€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 365,00		€ 0,00	€ 0,00	€ 0,00	€ 365,00
17	€ 320,00	€ 0,00	€ 0,00	€ 0,00	€ 260,50	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 580,50
18	€ 0,00	€ 0,00	€ 0,00		€ 0,00	€ 0,00	€ 480,25	€ 0,00	€ 0,00	€ 385,50	€ 0,00	€ 0,00	€ 865,75
19		€ 0,00	€ 515,75	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00		€ 0,00	€ 0,00	€ 345,00	€ 860,75
20	€ 0,00	€ 405,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 295,00	€ 0,00	€ 0,00		€ 700,00
21	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 525,50	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00		€ 525,50
22	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 380,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 430,75	€ 810,75
23	€ 0,00	€ 0,00	€ 0,00	€ 440,25	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 505,00	€ 0,00	€ 945,25
24	€ 285,50	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 285,50
25	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 355,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 355,00
26	€ 0,00	€ 0,00	€ 395,50	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 395,50
27	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 580,25	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 580,25
28	€ 0,00	€ 320,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00		€ 320,00
29	€ 0,00	€ 0,00		€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 475,00	€ 0,00		€ 475,00
30	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 365,00		€ 365,00
TOTALE	€ 2.001,00	€ 2.341,30	€ 2.182,05	€ 2.196,50	€ 1.877,05	€ 1.551,25	€ 2.077,00	€ 1.856,25	€ 1.896,40	€ 2.290,50	€ 2.372,30	€ 2.226,50	€ 24.868,10

Figura 4.4: Esempio di file tabulare.

```
info_encoded = '''{'file_info': {'name': 'fatture.xlsx', 'company':
    'Test_Company', 'sheets': ['fatture_mensili (1)'], 'total_sheets
    ': 1}, 'sheets_data': [{'sheet_name': 'fatture_mensili (1)', '
    rows': 31, 'columns': 14, 'column_names': ['Giorno', 'Gennaio',
    'Febbraio', 'Marzo', 'Aprile', 'Maggio', 'Giugno', 'Luglio', '
    Agosto', 'Settembre', 'Ottobre', 'Novembre', 'Dicembre', 'TOTALE
    '], 'sample_data': [{'Giorno': 1, 'Gennaio': 250.5, 'Febbraio':
    '[EMPTY]', 'Marzo': 180.0, 'Aprile': 0.0, 'Maggio': 420.75, '
    Giugno': 0.0, 'Luglio': 0.0, 'Agosto': 310.2, 'Settembre': 0.0,
    'Ottobre': 275.0, 'Novembre': 0.0, 'Dicembre': 195.5, 'TOTALE':
    1631.95}, {'Giorno': 2, 'Gennaio': '[EMPTY]', 'Febbraio': 315.8,
     'Marzo': 0.0, 'Aprile': 0.0, 'Maggio': 0.0, ... ... , '
    column_types': {'Giorno': 'object', 'Gennaio': 'float64', '
    Febbraio': 'float64', 'Marzo': 'float64', ... ... , 'Settembre':
     {'min': 0.0, 'max': 1896.4, 'non_null_count': 29}, 'Ottobre': {
    'min': 0.0, 'max': 2290.5, 'non_null_count': 31}, 'Novembre': {'
    min': 0.0, 'max': 2372.3, 'non_null_count': 31}, 'Dicembre': {'
    min': 0.0, 'max': 2226.5, 'non_null_count': 26}, 'TOTALE': {'min
    ': 285.5, 'max': 24868.1, 'non_null_count': 31}}}]}'''
```

Listing 4.1: Encoding file tabulari: esempio di stringa risultante.

Effettuato l'encoding, è ora possibile inviare a Claude i file:

- **File in formato PDF**: come presentato nel listing 4.2.

```
response = self.client.messages.create(
    model = self.claude_model,
    system = prompt_info,
    messages = [{
            "role": "user",
            "content": [
                {
                    "type": "document",
                    "source": {
                        "type": "base64",
                        "media_type": "application/pdf",
                        "data": file_encoded
                    }
                },
                {
                    "type": "text",
```

```
                        "text": prompt_info
                    }
                ]
            }],
        max_tokens = 20,
    )
```

Listing 4.2: Interfaccia Claude: invio file PDF.

- **Immagini**: come presentato nel listing 4.3, file nel formato *jpeg*, *png* e *gif* (il listing presentato è analogo a 4.2, tranne per la parte riportata).

```
    response = self.client.messages.create(
        ...
        {
        "type": "image",
        "source": {
            "type": "base64",
            "media_type": image_media_type,
            "data": file_encoded,
        },
        ...
    )
```

Listing 4.3: Interfaccia Claude: invio immagini.

Tutti gli altri formati di file devono essere inviati tramite **input prompt**: ovvero effettuare l'encoding dei file in formato stringa ed inserire il tutto nel prompt passato all'AI (come riportato nel listing 4.4).

```
response = self.client.messages.create(
    model = self.claude_model,
    system = prompt_info,
    messages = [{
            "role": "user",
            "content": prompt_info
        }],
    max_tokens=20,
)
```

Listing 4.4: Interfaccia Claude: invio file tramite prompt.

III. Classificazione documento ed estrazione informazioni

Claude analizza i documenti ricevuti: **assegna la categoria** di appartenenza al documento caricato (tra quelle definite dall'utente) ed **estrae le informazioni** necessarie per salvare i file secondo il *Models* definito nel listing 4.5.

Facendo riferimento al listing precedentemente enunciato, alcuni dei campi più rilevanti del Model sono: *binary_file_content* a riga 5 (che permette di salavare il file sotto forma di binario nel database), *file_hash* a riga 8 (necessario per controllare il possibile caricamento nel database di file duplicati), *mime_type* a riga 9 (necessario per permettere il download del file attraverso il client) e *upload_date* a riga 11 (data di caricamento del file nel database).

```
1  class DocumentModel(Document):
2      # required document fields
3      filename = StringField(required=True, max_length=255)
4      file_extension = StringField(required=True, max_length=10)
5      binary_file_content = BinaryField(required=True)
6      file_size = IntField(required=True)
7      # to prevent duplicate uploads
8      file_hash = StringField(required=True, unique=True)
9      mime_type = StringField(required=True)
10     file_category = StringField(required=True)
11     upload_date = StringField(required=True)
12
13     # other optional document fields
14     sender = StringField()
15     receiver = StringField()
16     subject = StringField()
17     total_cost = FloatField()
18     document_date = StringField()
19
20     meta = {
21         # collection name in MongoDB
22         'collection': 'documents',
23         # internal indexing based on...
24         'indexes': ['filename', 'upload_date', 'file_hash']
25     }
```

Listing 4.5: Document Model.

IV. Elaborazione della query ed aggiornamento del *database context*

Poiché il servizio sviluppato si basa sul *Model Context Protocol*, per la creazione delle query (che permettono di salvare i documenti secondo i model enunciati precedentemente), l'AI necessita del *contesto del database*. Nel progetto sviluppato il contesto è presente nella cartella */LOGFILE* presente in locale.

La cartella */LOGFILE* è designata ai *logs* di progetto. Nello specifico esistono tre tipologie di logs salvati in questa cartella:

- *Info logs*: questo file è dedicato al tracciamento delle operazioni, interne ed esterne, eseguite dal server.
- *Error/Warning logs*: in questo file vengono segnati tutti gli errori critici che avvengono durante lo svolgimento delle varie operazioni.
- *Schema logs*: questo file tiene traccia del *contesto* da fornire a Claude, ovvero lo schema interno al database (questo file viene aggiornato ogni qualvolta si eseguono operazioni che cambiano la struttura delle collezioni interne al database come aggiungere o rimuovere un file).

Sia nel file di *Info* che in quello di *Error/Warning*, i logs presenti sono accompagnati dal *timestamp* nella forma *'YYYY-mm-dd H:M:S - explaination'*: questo risulta particolarmente utile in un contesto *production aziendale*. Permette infatti di risolvere un possibile errore comprendendo quando temporalmente (ed in che punto del codice) è sorto il problema.

V. Salvataggio dei documenti nel database

Per il salvataggio e la manipolazione dei file documentali, si usufruisce delle funzioni presenti in *doc_ category_ operations.py* e *utils.py*.

Le funzioni presenti nel primo file sono state implementate segendo la filosofia CRUD: la funzione `upload_to_db()` crea l'istanza del documento nel database, la funzione `execute_query()` permette l'esecuzione della query creata dall'AI in formato `__raw__` (query ricavata attraverso `create_db_query()` presente nel file claude_query_creator.py del modulo /ai_microservices). Infine attraverso la funzione `delete_documents()` è possibile eliminare documenti secondo la query definita da AI tramite `create_db_query()`.

Sono inoltre presenti due funzioni di supporto: `count_documents()`, per il conteggio dei documenti presenti nel database, e `__safe_query_parse()` per il parsing delle stringhe a for-

mato JSON. Poiché l'AI restituisce le query in formato *string*, l'azione di parsing risulta essere necessaria per poter eseguire le query in formato `__raw__` su MongoDB).

In *utils.py* è presente la funzione di supporto `check_file_in_db()`: la funzione verifica la presenza del documento nel database, attraverso il confronto tra il codice *hash* del documento caricato nel programma e quello dei documenti già presenti nel database (listing 4.6).

```
def check_file_in_db(self, file_path: Path) -> bool:
    try:
        with open(file_path, 'rb') as f:
            binary_file_content = f.read()
        file_hash = hashlib.md5(binary_file_content).hexdigest()
        if DocumentModel.objects(file_hash=file_hash).first():
            self.logger.write_info_in_log_file(f"(WARNING) File already
                exists in DB: {file_path.name}")
            return True
        return False
    except Exception as e:
        self.logger.write_error_in_log_file(f"Error checking file in DB: {e}
            ")
        return False
```

Listing 4.6: Controllo presenza file nel database mediante confronto codice hashing.

Questa funzione risulta particolarmente utile per evitare che l'utente carichi nel database copie dello stesso file, prevenendo il degrado delle performance nel lungo termine.

VI. Feedback verso l'utente

Infine viene fornito all'utente un messaggio di feedback: viene restituito un messaggio di successo in caso di operazione conseguita correttamente, altrimenti viene ritornato un messaggio di errore.

4.2.2 Richiesta di informazioni, o documenti, tramite *chat*

Nella figura 4.5 viene riportato, tramite diagramma flowchart, il flusso di operazioni che vengono effettuate ogni qualvolta l'utente richiede, attraverso la *chat-AI*, delle informazioni presenti sui file precedentemente salvati nel database oppure la restituzione di documenti specifici.

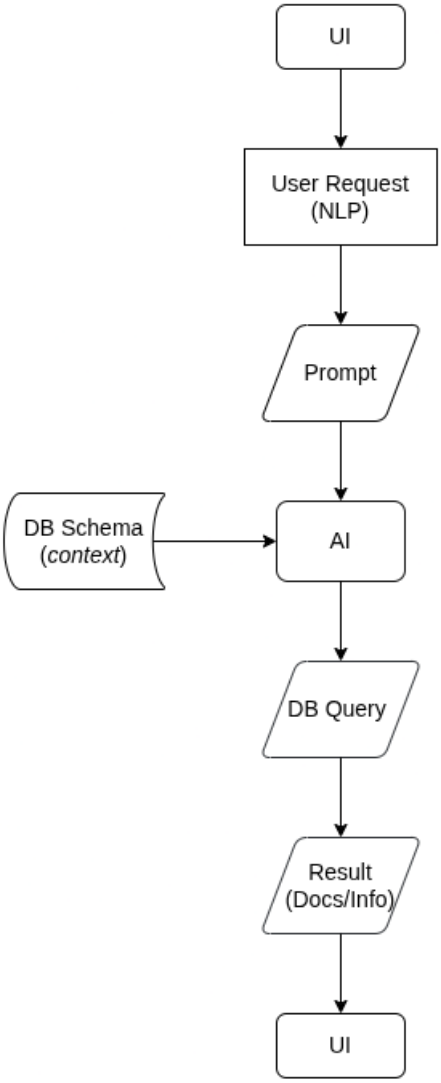

Figura 4.5: Flowchart: richiesta di informazioni, o documenti, tramite chat.

I. L'utente effettua la richiesta

L'utente, tramite *chat-AI*, effettua la richiesta di informazioni (o documenti) in linguaggio naturale. Si richiama quindi l'API **@router.get("/ask-docs-info")** per soddisfare la richiesta.

II. Elaborazione della query di ricerca

Analogamente a quanto espresso nel punto *IV* della sottosezione precedente, l'AI elabora la query tramite lo *schema log_file* al fine di trovare i file con le informazioni richieste dall'utente.

III. Raccolta file ed elaborazione risposta

Mediante l'uso delle funzioni apposite (spiegate nella sottosezione 4.3.2), viene elaborata la risposta da ritornare all'utente in linguaggio naturale.

IV. Restituzione risposta NLP e dei file richiesti

Infine viene ritornata, tramite chat, la risposta elaborata dall'AI. Inoltre, attraverso l'API **@router.get("/download-file/file_id")** (che permette di scaricare il file, dato il suo *file_id*, tramite codifica binaria), vengono forniti per il download i documenti ricavati tramite la query elaborata da Claude.

4.2.3 Operazioni di manipolazione delle *categorie documentali*

Oltre a poter caricare e chiedere informazioni al programma (espresse nella sezione precedente mediante flowcharts), attraverso la sezione */settings* del Client, l'utente può salvare, aggiungere od eliminare le *categorie dei documenti*.

Creare nuove categorie

Attraverso l'API **@router.post("/create-category")** è possibile creare nuove categorie nel database.

Prima di creare una o più categorie viene controllato, attraverso `str_exists_in_db()` (presente nel file *doc_category_operations.py* nella cartella */controllers*), se i dati riguardanti le categorie siano presenti o meno nel database.

Se già sono presenti dei dati si nega l'operazione per evitare conflitti interni, altrimenti si salva la stringa contenente le categorie secondo il *Model* espresso nel listing 4.7.

```
class FileCategory(Document):
    file_category_string = StringField(required=True)
    full_file_category_and_descr_string = StringField(required=True)

    meta = {
        'collection': 'file_categories',
        'indexes': ['full_file_category_and_descr_string']
    }
```

Listing 4.7: Document category Model.

Aggiungere categorie a quelle già esistenti

Per aggiungere nuove categorie a quelle già presenti nel database, è possibile usufruire dell'API **@router.put("/update-category")** (che richiama `update_file_category_str()`).

Nello specifico prima si ottiene la stringa con la lista completa delle categorie salvate, si elimina e successivamente si salva una nuova lista dove sono presenti anche le nuove categorie aggiunte dall'utente.

Eliminare le categorie

Mediante apposita interfaccia, è possibile eliminare tutte le categorie presenti nel database richiamando l'API **@router.delete("/delete-all-categories")**.

4.3 Interazione con i servizi di Claude

Nel progetto di tesi proposto, parte fondamentale è il modulo dedicato alle interazioni con i servizi offerti da Claude.

In questa sezione si analizza nello specifico la parte di progetto che implementa le interfacce di interazione con Claude, evidenziandone l'utilità ed analizzando l'implementazione.

4.3.1 Formulazione del prompt

Prima di analizzare dal punto di vista implementativo le interfacce usate, è necessario esaminare l'elemento che permette di guidare l'AI nello svolgimento dei compiti assegnati: il *prompt*.

I *prompt* sono istruzioni che, assegnato un determinato compito, vengono fornite all'AI per comprendere il contesto specifico in cui operare.

Attualmente, nel progetto presentato, i prompt forniti all'AI sono stati redatti *ad hoc* in base al contesto d'utilizzo degli strumenti di analisi automatizzata. In particolare, sono stati definiti prompt specifici per i seguenti compiti: **analisi dei dati**, **estrazione delle informazioni rilevanti** dai file analizzati, **classificazione dei documenti**, **creazione delle query** (eseguibili sul database) e **aggiornamento del contesto locale** (ossia dello schema interno del database), oltre che per **l'interpretazione delle richieste in linguaggio naturale dell'utente** e la **formulazione di risposte testuali** da restituire tramite chat.

Nonostante i prompt forniti all'AI siano stati scritti in modo specifico in funzione del compito da eseguire, seguono generalmente la seguente struttura:

- **Direttive d'esecuzione**: vengono inserite l'insieme delle informazioni necessarie affinché l'AI capisca quale è il compito assegnato (un esempio è presente nel listing 4.8).

```
prompt_info = "Sei un assistente AI che aiuta a tenere traccia delle
    informazioni che sono archiviate in un database non relazionale
    (MongoDB). ..."
```

Listing 4.8: Prompt: esempio di direttive d'esecuzione.

- **Oggetto di lavoro**: vengono fornite all'AI le informazioni su cui deve eseguire il compito assegnato (come nel listing 4.9).

```
prompt_info = prompt_info + f"... Le informazioni che devi
    interpretare sono le seguenti: {info}, e la richiesta dell'
    utente e': {user_request} ..."
```

Listing 4.9: Prompt: esempio di regole d'esecuzione.

- **Regole d'esecuzione**: ovvero tutte le regole che l'AI deve rispettare durante l'esecuzione del compito assegnato (come listing 4.10).

```
prompt_info = prompt_info + "... aggiorna la struttura del database.
    NON ritornare MAI una spegazione."
```

Listing 4.10: Prompt: esempio di regole d'esecuzione.

Nella scrittura del prompt bisogna trovare un equilibrio tra specificità e generalità di linguaggio: si riscontra infatti dai vari test che sia un prompt troppo generale, che uno troppo verboso, causino un perdita di performance da parte dell'AI nella comprensione del prompt stesso. Ciò comporta la generazione di risposte indesiderate che non rispecchiano le direttive indicate nel prompt e che quindi non soddisfano a pieno le richieste dell'utente.

4.3.2 Implementazione interfacce dei servizi AI

Nella cartella */ai_microservices* sono presenti tutti i file che permettono di interfacciarsi con i servizi offerti da Claude AI.

La struttura delle funzioni, che usano l'interfaccia di accesso all'API di Claude, seguono la forma:

I **Operazioni pre-interfaccia di Claude**: comprendono tutte le operazioni (come apertura ed *encoding* di file) svolte prima di usufruire dell'interfaccia di accesso ai servizi di Claude.

II **Definizione del prompt**: si forniscono le direttive sulle operazioni che l'AI deve eseguire.

III **Accesso all'interfaccia di Claude**: sul modello definito nel listing 2.1.

IV **Restituzione risposta di Claude**: viene restituita la risposta di Claude, riguardante il compito assegnato, attraverso stringhe in linguaggio naturale.

Di seguito vengono riportate le principali funzionalità presenti nelle interfacce dei file contenuti nella cartella */ai_ microservices*.

Interfaccia per l'elaborazione del *database context*

Attraverso il file *claude_ db_ contexter.py*, il programma tiene traccia del *contesto del database*: nello specifico tiene annotato lo *schema* interno al database, aggiornando localmente lo *schema logfile* presente nella cartella */LOGFILE* durante le operazioni che riguardano l'aggiunta, o la rimozione, di file da MongoDB (rispettivamente tramite le funzioni `create_db_context()` e `delete_in_db_context()`).

Interfaccia per il *processing* e salvataggio del file nel database

Nel file *claude_ doc_ classificator.py*, sono presenti funzioni che processano i documenti caricati dall'utente prima che vengano salvati nel database. Nello specifico tramite la funzione `preprocess_file()`, si processa il file come indicato dal listing 4.11:

1. Si controlla se il file da analizzare sia già presente in database attraverso il **confronto tra hashing** del file caricato e quello dei file presenti nel database (in tal caso si salta l'intero passaggio di analisi, e salvataggio, del file come mostrato a righe 8-9 del listing).

2. Si **ottiene nome ed estensione** del file, necessario per capire quale tipologia di encoding effettuare (riga 12).

3. Si effettua l'apposito **encoding** e si fa **processare il file a Claude**. In specifico si eseguono le operazioni di: *classificazione del documento* (riga 17 del listing), *estrazione dei dati rilevanti* (riga 20 del listing), *scrittura dello schema log* e *creazione della query* da eseguire sul database (riga 22 del listing).

4. Si **crea una copia** del file e si **salva nel database** (riga 25).

5. Si **elimina la copia** del file presente in locale sulla cartella */DUMP*.

```
def preprocess_file(self) -> None:
    # start timer estimation (benchmark uses)
    self.time_estimator.start_counting_time()

    for item in self.path_DUMP.iterdir():
        if self.__is_file_considerable(item):
            # skip analysis if already in db
            if self.utils.check_file_in_db(item):
                continue

            #1st obtain file extension type
            file_name, file_extension = os.path.splitext(item.name)
            self.logger.write_info_in_log_file(f"(at func: preprocess_file)
                Processing file: {file_name} with extension {file_extension}"
                )

            #2nd get classification from claude based on its extension
            company_name = "company_name"
            claude_classification_response, file_encoded = self.
                __analyze_file(item, file_extension, company_name)
            self.logger.write_info_in_log_file(f"(at func: preprocess_file)
                Claude's response about {company_name} document: {
                claude_classification_response}")

            claude_info_extracted, exact_time = self.claude_extractor.
                claude_extract_info(item, file_encoded,
                claude_classification_response)
            # create/update the db context with the new info extracted
            self.db_context_creator.create_db_context(claude_info_extracted)

            #3rd upload file to DB
            upload_code = self.file_ops.upload_to_db(item,
                claude_classification_response, claude_info_extracted,
                exact_time)

            #4th delete file in DUMP folder
            if upload_code == 1:
                item.unlink(missing_ok=True)
                self.logger.write_info_in_log_file(f"(at func:
                    preprocess_file) Deleted file {item.name} from DUMP
```

```
                    folder after successful upload.")
31              else:
32                  self.logger.write_warning_in_log_file(f"(at func:
                        preprocess_file) File {item.name} not deleted from DUMP
                        folder due to upload error.")
33
34              # update total operations counter (benchmark uses)
35              self.time_estimator.total_operations += 1
36
37
38      # stop timer estimation (benchmark uses)
39          elapsed_time, avg_time_per_op = self.time_estimator.
                estimate_total_time_and_op()
40          self.logger.write_info_in_log_file(f"(at func: preprocess_file)
                Total elapsed time: {elapsed_time}, Average time per operation: {
                avg_time_per_op}")
```

Listing 4.11: Processing del file caricato dall'utente.

Interfaccia per l'estrazione delle informazioni rilevanti dai file caricati

Nel file *claude_info_extractor.py* sono presenti le funzioni necessarie all'estrazione dei dati rilevanti dai file caricati dell'utente. Nello specifico, seguendo il modello secondo cui vengono salvati i documenti nel database, si estraggono: *mittente documento*, *destinatario documento*, *oggetto documento*, *costo totale* riportato nel documento (parametro pensato specialmente in ottica di caricamento di documenti aziendali e finanziari) e *data documento* (e.g. data di invio di una email).

Inoltre è presente anche la funzione `post_process_extraction()` che permette il corretto svolgimento dell'operazione di aggiornamento dello *schema log* controllando la coerenza tra lo schema generato dai dati estratti dal documento caricato e lo *schema* interno al database.

Interfaccia per l'elaborazione del linguaggio naturale

Nel file *claude_interpreter.py* sono presenti le funzioni, che tramite accesso all'interfaccia di Claude, forniscono supporto all'interpretazione delle richieste dell'utente (effettuate in linguaggio naturale) espresse nella *chat-AI*.

La funzione `wants_file()` permette di comprendere se l'utente ha espressamente richiesto di ritornargli documenti presenti nel database, oppure se ha semplicemente espresso una richiesta di informazioni sui documenti che ha precedentemente salvato.

La funzione `interpret_response()` permette la formulazione di risposte in linguaggio naturale per le richieste espresse dall'utente: forniti i dati, ricavati dal database attraverso query elaborate da Claude, viene creata una risposta coerente secondo quanto richiesto da ritornare all'utente tramite *chat-AI*.

Infine le funzioni `interpret_file_categories()` e `interpret_file_descr()` forniscono rispettivamente: supporto alla comprensione dell'appellativo della categoria ed assistenza alla comprensione della descrizione della categoria, (ogni qualvolta vengono definite dall'utente, tramite interfaccia apposita, delle nuove categorie documentali).

Interfaccia per la creazione di query

Attravreso la funzione `create_db_query()` (presente in *claude_ query_ creator.py*) è possibile, dato lo schema del database (ricavato dal file di log in locale) e data la richiesta dell'utente in linguaggio naturale, creare le query da eseguire sul database; permettendo quindi di ricavare i documenti da cui vengono estratte le informazioni richieste dall'utente.

L'esecuzione diretta della query è possibile grazie alla libreria *mongoengine*: garantisce infatti di eseguire le query sotto forma di stringa (elaborate dall'AI) direttamente sul'*engine* del database tramite il comando riportato nel listing 4.12.

```
1 documents = DocumentModel.objects(__raw__=query)
```

Listing 4.12: Esecuzione query elaborata da AI direttamente sul database.

4.3.3 Scelta del modello di Claude

Facendo riferimento ai modelli offerti da Anthropic in [4], il programma presentato in questo progetto di tesi utilizza il modello denominato *Haiku* (questo modello è pensato principalmente per compiti ripetitivi e specifici, dove si predilige la velocità rispetto alla correttezza assoluta della risposta).

Nonostante si possano ritenere soddisfacenti i risultati ottenuti mediante l'uso di *Haiku* come modello di AI, il modello riscontra difficoltà nell'interpretare correttamente frasi formulate con un linguaggio naturale complesso: ciò causa la formulazione di query che portano risultati inconsistenti nella ricerca dei dati nel database (nonostante la presenza effettiva dei dati richiesti nel database). Un esempio ricorrente, in cui l'AI crea query inconsistenti per la ricerca, è il caso in cui l'utente richieda documenti, o informazioni, esplicitando nella richiesta determinati intervalli di tempo.

Per i test eseguiti durante la fase di sviluppo si riscontra che la creazione approssimativa di query, da parte del servizio di AI, accade all'incirca il 20/30% delle volte.

Per linguaggio naturale complesso, dove spesso si può esprimere lo stesso concetto complicando inutilmente il quesito (come può avvenire con la lingua italiana), è consigliabile, in un contesto aziendale, l'uso di un modello che permette un ragionamento sulle informazioni più prolungato e dettagliato (come avviene per i modelli *Sonnet* e *Opus*).

In conclusione, come anche descritto nella sezione 2.3.1, l'uso del modello *Sonnet* (dove si ha un giusto compromesso tra complessità ragionamento, velocità di risposta e costi di utilizzo) si presuppone essere particolarmente indicato per questa tipologia di progetto.

4.4 Funzionalità di supporto del programma

Nella cartella */utils* sono presenti i file che supportano il programma nello svolgimento di compiti secondari ma necessari. Un esempio è il conteggio dei token nelle richieste a Claude oppure il sistema di *logging* delle operazioni.

Di seguito si riporta la lista esaustiva delle funzionalità di supporto presenti che assisto il programma nelle operazioni principali.

Benchmarker delle prestazioni di caricamento file

Il file *benchmarker.py* è dedicato alle operazioni di benchmarking delle procedure di caricamento dei file nel database. Nello specifico, attraverso le relative funzioni, calcola il tempo medio di caricamento di un file, includendo anche il tempo necessario per la sua elaborazione da parte dell'AI (ovvvero includendo il tempo trascorso per le operazioni di *estrazione di dati rilevanti* e *classificazione documento*).

Pulitore cartella /DUMP

Il file *cleaner_tool.py* è un semplice *script* che serve a pulire la directory */DUMP* dai vari documenti presenti al suo interno.

Risulta particolarmente utile in contesto di sviluppo: se avviene un errore critico durante il processo di caricamento del file, prima che questo venga salvato nel database, è possibile ripulire la directory evitando la nascita di incongruenze tra servizio di AI (per quanto riguarda lo *schema di contesto* per l'AI) e lo stato interno al database.

Sistema personalizzato di logging

In *error_logger.py* sono presenti tutte le funzioni relative alla creazione, manipolazione e scrittura dei file di log (*schema_log*, *info_log* e *error_log*) nella cartella */LOGFILE*.

Per evitare di appesantire il servizio con file di log di dimensioni eccessive, attraverso le apposite funzioni, viene effettuato un controllo automatico sugli *info_log* ed *error_log*: i file vengono eliminati se risultano più vecchi di 2 giorni o se risultano di dimensioni maggiore o uguale a 1 Mb.

Calcolatore di *token*

Il file *ocr.py*, attraverso la funzione `anthropic_token_estimator()`, permette di calcolare in modo preciso il numero di token presenti nella richiesta rivolta a Claude, attraverso l'uso dell'interfaccia apposita (secondo quanto indicato nella documentazione ufficiale [6]).

Questa funzionalità risulta particolarmente utile in un contesto aziendale per il calcolo delle spese sostenute dal programma durante il processing dei documenti caricati dall'utente.

Seguendo quanto riportato nella sottosezione 4.2.1, il calcolo dei token è specifico per singola tipologia di file caricato (come riportato nel listing 4.13).

```
def anthropic_token_estimator(self, api_client: any, file_name: str,
    encoded_file: str, claude_model: str, prompt: str) -> None:
     try:
         if file_name.endswith('.pdf'):
             response = api_client.messages.count_tokens(
                 model=claude_model,
                 messages=[{
                     "role": "user",
                     "content": [
                         {
                             "type": "document",
                             "source": {
                                 "type": "base64",
                                 "media_type": "application/pdf",
                                 "data": encoded_file
                             }
                         },
                         {
                             "type": "text",
                             "text": prompt
                         }
```

```
                    ]
                }]
            )

        elif file_name.endswith('.jpg') or file_name.endswith('.jpeg') or
            file_name.endswith('.png') or file_name.endswith('.gif'):

            if file_name.endswith('.jpg'):
                image_media_type = 'image/jpeg'
            elif file_name.endswith('.jpeg'):
                image_media_type = 'image/jpeg'
            elif file_name.endswith('.png'):
                image_media_type = 'image/png'
            elif file_name.endswith('.gif'):
                image_media_type = 'image/gif'

            response = api_client.messages.count_tokens(
                model=claude_model,
                messages=[{
                    "role": "user",
                    "content": [
                        {
                            "type": "image",
                            "source": {
                                "type": "base64",
                                "media_type": image_media_type,
                                "data": encoded_file
                            }
                        },
                        {
                            "type": "text",
                            "text": prompt
                        }
                    ]
                }]
            )

        # all tokens are in prompt! (in :str format)
        else:
            response = api_client.messages.count_tokens(
```

```
60                 model=claude_model,
61                 messages=[{
62                     "role": "user",
63                     "content": prompt
64                 }]
65             )
66
67         self.logger.write_info_in_log_file(f"Estimated tokens by Anthropic
               API: {response.input_tokens}")
68     except Exception as e:
69         self.logger.write_warning_in_log_file(f"(at func:
               anthropic_token_estimator) Error estimating tokens with Anthropic
                API: {e}")
```

Listing 4.13: Procedura di calcolo dei token per la richiesta effettuata a Claude.

Nel calcolo dei costi d'utilizzo, oltre al numero complessivo di token utilizzati nella richiesta, è importante segnalare che assume rilievo anche il modello di Claude impiegato: il costo dell'uso del modello più complesso può infatti risultare in una spesa fino a venti volte superiore rispetto a quella legata all'uso del modello base.

Capitolo 5

Conclusioni

Nel seguente capitolo si riassume il contenuto esposto in questa tesi, evidenziando gli attuali limiti architetturali e possibili sviluppi del progetto presentato.

5.1 Limiti architetturali del progetto

In questa sezione si discutono gli attuali limiti architetturali e vengono proposte soluzioni attuabili al fine di arginare i problemi evidenziati.

5.1.1 Limiti intrinsechi al progetto e proposte di soluzioni

Il progetto presentato in questa tesi non presenta dei veri e propri limiti che generano problemi nell'esecuzione delle operazioni.

Nonostante ciò di seguito vengono riportati problematiche, che sono attualmente presenti nel codice di progetto, che impediscono di considerare programma (da un punto di vista aziendale) pronto per il rilascio come servizio usufruibile da una vastità di utenti differenti.

Modello documentale

Il primo limite architetturale che viene discusso è il modello secondo cui vengono salvati i file caricati dall'utente nel database.

Facendo riferimento al listing 4.5, i dati che vengono estratti (e salvati) dai documenti sono solo: *sender (mittente), receiver (destinatario), subject (oggetto), total_cost (costo totale riportanto nel documento) e document_date (data segnata all'interno del documento).* Ciò implica che, pur essendo il sistema orientato al salvataggio di file con contenuto eterogeneo (grazie alla capacità di lasciare la completa libertà all'utente sulla tipologia di documenti che possono essere

riconosciuti, tramite definizione della *file_ category* in forma *'NOME:definizione'*), attualmente è presente un vincolo strutturale sulla tipologia di documenti salvabili nel database, rendendo consigliabile il caricamento solo di documenti aziendali e finanziari (come fatture, bolle o documenti di trasporto).

Un altro problema che sorge, come diretta conseguenza del limite esposto, è l'assenza di generalità assoluta sulle informazioni che è possibile chiedere a Claude. Si prenda come esempio la domanda dell'utente *'Indicami tutti i nomi propri scritti sull'ultima fattura che ti ho caricato'*: il servizio di AI non riesce a soddisfare la richiesta perché sullo *schema log_file* non è presente alcun campo (e quindi neanche nei modelli interni al database) che riesce a soddisfare la domanda posta.

Una possibile **soluzione** per arginare il problema consiste **nell'aggiunta di un campo generico**: l'idea è quella di introdurre un campo (ad esempio *'relevant_ info'*) in cui venga salvata, sotto forma di stringa, la raccolta delle informazioni che l'AI ritiene rilevanti rispetto al documento analizzato.

Inconsistenza generazione automatizzata delle query

Il secondo limite che viene analizzato è l'inconsistenza nelle risposte restituite dell'AI, ovvero la restituzione di una risposta non coerente o non pertinente rispetto al compito assegnato al servizio di intelligenza artificiale. Ciò è principalmente causato dal fraintendimento, da parte dell'AI, della richiesta ricevuta, basando l'intero ragionamento su premesse errate e quindi causando la generazione di una risposta inattesa.

Come già accennato nella sezione 4.3.3, in questo progetto di tesi la possibilità di ottenere risposte inconsistenti da parte dell'AI è riscontrabile nella porzione di servizio che gestisce la creazione di *query* per la ricerca dei documenti presenti nel database. Se infatti l'utente pone, nella sezione dedicata alla *chat-AI*, domande in linguaggio naturale complesso (ad esempio domande lunghe, verbose o tramite termini generali che presentano molteplici significati) può capitare che il servizio di *query_ generator* elabori una risposta che non riesca a individuare, nel database, i file necessari al fine di soddisfare la richiesta stessa (nonostante ci siano le premesse per trovare i file target e soddisfare quindi le richieste dell'utente).

Per cercare di arginare il problema, si propone come **soluzione** l'uso di un **modello con capacità di ragionamento maggiori**: l'idea è quindi usare modelli di AI, come *Claude Sonnet* o *Claude Opus*, come base per la porzione di programma che usufruisce dei servizi di AI; oppure usare come sistema di AI un modello *in locale* specificamente allenato sul riconoscere

le informazioni rilevanti dai documenti presentati e sulla formulazione di *query* che selezionano in modo accurato le informazioni da estrarre dal database.

Sia la soluzione proposta ora, che la soluzione relativa ai limiti dell'attuale Model, necessitano di testing per verificare l'effettivo miglioramento delle prestazioni (per quanto concerne la creazione delle *query target*) e sulla migliore generalità d'applicazione del servizio sviluppato.

5.2 Lavori simili

Grazie al crescente interesse per lo sviluppo di strumenti di AI, l'ambito della gestione documentale automatizzata risulta sempre più appetibile per aziende e privati. Come conseguenza di ciò, negli ultimi anni, vengono sviluppati sempre più servizi per la gestione di file che integrano l'intelligenza artificiale al fine di rendere le operazioni di ricerca più semplici ed efficienti.

Alcuni esempi di noti servizi per la gestione documentale, che integrano l'AI, sono:

- *M-Files*: sistema di gestione documentale che automatizza i processi utilizzando l'AI, e i metadata dei file, per trovare facilmente le informazioni richieste, memorizzando nativamente i contenuti all'interno della piattaforma Microsoft 365 [26].

- *Google Document AI*: elaboratore di documenti che automatizza l'estrazione delle informazioni dai documenti strutturati [23].

- *DocuWare IDP*: sistema di gestione documentale con capacità di riconoscimento automatico del testo da documenti strutturati. Offre inoltre la possibilità di eseguire operazioni di splitting, cropping e classificazione direttamente sui documenti [19].

Il *Model Context Protocol* rappresenta un ambito di ricerca ancora recente e in larga parte da esplorare. Nonostante l'interesse crescente da parte del mondo accademico e delle grandi aziende, sono tuttora limitati i lavori che affrontano tematiche strettamente analoghe a quelle sviluppate nel progetto di tesi presentato.

Si osserva tuttavia un progressivo aumento di pubblicazioni che ne favoriscono l'integrazione in un numero sempre maggiore di contesti applicativi. Alcuni tra gli esempi di implementazione di *server MCP* attualmente presenti sul mercato sono:

- Il server MCP per il controllo del *workflow in GitHub* ([22]): connette strumenti AI direttamente alla piattaforma GitHub, fornendo ad *agenti AI* la possibilità di analizzare ed automatizzare il processo di *repository management*.

- Il server MCP per il controllo della *containerizzazione in Docker* ([17]): il server MCP, che si interfaccia con le API di Docker Hub, rende accessibili agli *LLM* le repository di Docker, facilitando i processi di creazione dei *container*.

- Il server MCP per la gestione dei *file in database in Azure* ([28]): permette ad applicazioni e modelli AI di comunicare con i dati ospitati in Azure (per il database relazionale PostgreSQL), permettendo ai modelli di AI l'accesso e l'elaborazione dei dati presenti nelle tabelle.

5.3 Considerazioni finali e lavori futuri

Il progetto presentato in questa tesi presenta basi solide per la creazione di un applicativo complesso, con la possibilità di applicazione (aziendale e non) per casi di necessità reale.

Nonostante ciò, il programma sviluppato risulta ancora grezzo e quindi necessita ancora di lavoro prima di renderlo presentabile come servizio offerto ad un pubblico vasto, in primo luogo risolvendo i problemi esposti nella sezione 5.1.

Inoltre, ulteriori margini di miglioramento riguardano sia la *User Interface* che la gestione interna degli utenti. Al momento, il servizio è progettato per un'installazione in *self-hosting* destinata all'uso di un singolo utente; tuttavia, in una prospettiva aziendale orientata all'erogazione come servizio *SaaS*, risulterebbe più appropriato l'hosting su un'infrastruttura centralizzata (ad esempio su AWS), che consenta l'accesso a più utenti contemporaneamente. Tale miglioria del programma richiede l'implementazione di un sistema di *gestione multi-utente*.

Bibliografia

[1] Alex Rossi (Axelredx). *Bechelor Thesis Project*. https://github.com/Axelredx/Bechelor-Thesis-Project. Accessed: 2025-10-09. 2025.

[2] Anthropic. *Anthropic Python API library*. https://pypi.org/project/anthropic/. Accessed: 2025-10-03. 2025.

[3] Anthropic. *Claude API Documentation*. Accessed: 2025-10-01. 2025. URL: https://docs.claude.com.

[4] Anthropic. *Claude Models Documentation*. Accessed: 2025-10-02. 2025. URL: https://docs.claude.com/en/docs/about-claude/models/overview.

[5] Anthropic. *Claude Support Documentation*. Accessed: 2025-10-02. 2025. URL: https://docs.claude.com/en/docs/build-with-claude/pdf-support.

[6] Anthropic. *Claude Token-Counting Documentation*. Accessed: 2025-10-02. 2025. URL: https://docs.claude.com/en/docs/build-with-claude/token-counting.

[7] Inc. Anthropic. *Anthropic Console — Login*. https://console.anthropic.com/login. Accessed: 2025-10-09.

[8] Axios. *Getting Started | Axios Docs*. Accessed: 2025-10-04. 2025. URL: https://axios-http.com/docs/intro.

[9] Steve Canny. *Create, read, and update Microsoft Word .docx files*. https://pypi.org/project/python-docx/. Accessed: 2025-10-03. 2025.

[10] Tom Christie. *The lightning-fast ASGI server.* https://pypi.org/project/uvicorn/. Accessed: 2025-10-03. 2025.

[11] Jeffrey A. Clark. *Python Imaging Library*. https://pypi.org/project/pillow/. Accessed: 2025-10-03. 2025.

[12] Samuel Colvin. *Data validation using Python type hints*. https://pypi.org/project/pydantic/. Accessed: 2025-10-03. 2025.

[13] Wikipedia contributors. *MongoDB — Wikipedia*. https://en.wikipedia.org/wiki/MongoDB. Last modified: 2015-07-17; Accessed: 2025-10-08. 2025.

[14] Datadog Security Labs. *Container security fundamentals part 2: Isolation & namespaces*. Accessed: 2025-10-04. 2023. URL: https://securitylabs.datadoghq.com/articles/container-security-fundamentals-part-2/.

[15] Matthew Walker Destiny Peterson. *Extracts emails and attachments saved in Microsoft Outlook's .msg files*. https://pypi.org/project/extract-msg/. Accessed: 2025-10-03. 2025.

[16] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[17] Docker Inc. «Introducing Docker Hub MCP Server». In: *Docker Blog* (2025). Accessed: 2025-10-23. URL: https://www.docker.com/blog/introducing-docker-hub-mcp-server/.

[18] Docker Inc. *Security*. Docker Documentation. Accessed: 2025-10-04. 2025. URL: https://docs.docker.com/engine/security/.

[19] DocuWare GmbH. *DocuWare Introduces Intelligent Document Processing (DocuWare IDP)*. Press Release. Accessed: 2025-11-04. Dic. 2024. URL: https://start.docuware.com/press-center/docuware-introduces-intelligent-document-processing-docuware-idp.

[20] Mathieu Fenniak. *A pure-python PDF library capable of splitting, merging, cropping, and transforming PDF files*. https://pypi.org/project/pypdf/. Accessed: 2025-10-03. 2025.

[21] GitHub. *Evan You - README Stories*. Accessed: 2025-10-03. 2024. URL: https://github.com/readme/stories/evan-you.

[22] GitHub Inc. *GitHub MCP Server - Official Implementation*. https://github.com/github/github-mcp-server. Accessed: 2025-10-23. 2025.

[23] Google Cloud. *Document AI | Google Cloud*. Accessed: 2025-11-04. 2025. URL: https://cloud.google.com/document-ai.

[24] Xinyi Hou et al. «Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions». In: *ArXiv* abs/2503.23278 (2025). DOI: 10.48550/arXiv.2503.23278.

[25] Saurabh Kumar. *Read key-value pairs from a .env file and set them as environment variables*. https://pypi.org/project/python-dotenv/. Accessed: 2025-10-03. 2025.

[26] M-Files Corporation. *M-Files - Document Management System (DMS) with Workflow Automation*. Accessed: 2025-11-04. 2024. URL: https://www.m-files.com/.

[27] Harry Marr. *MongoEngine is a Python Object-Document Mapper for working with MongoDB.* https://pypi.org/project/mongoengine/. Accessed: 2025-10-03. 2024.

[28] Microsoft Azure. «Introducing Model Context Protocol (MCP) Server for Azure Database for PostgreSQL (Preview)». In: *Microsoft Community Hub* (2025). Accessed: 2025-10-23. URL: https://techcommunity.microsoft.com/blog/adforpostgresql/introducing-model-context-protocol-mcp-server-for-azure-database-for-postgresql-/4404360.

[29] Microsoft Learn. *Vue su Windows*. Accessed: 2025-10-03. 2025. URL: https://learn.microsoft.com/it-it/windows/dev-environment/javascript/vue-overview.

[30] Inc. MongoDB. *MongoDB Atlas: Piattaforma dati multi-cloud per sviluppatori*. https://www.mongodb.com/it-it/products/platform. Accessed: 2025-10-09. 2025.

[31] Inc. MongoDB. *MongoDB Documentation*. https://www.mongodb.com/docs/. Accessed: 2025-10-08.

[32] Manas Patil e Virag Lokhande. «Model Context Protocol (MCP): Enabling Scalable AI Data Integration». In: *International Journal For Multidisciplinary Research* (2025). DOI: 10.36948/ijfmr.2025.v07i02.43583.

[33] Python Software Foundation. *base64 — Base16, Base32, Base64, Base85 Data Encodings*. Python Standard Library Documentation, Accessed: 2025-10-03. Python Software Foundation. 2025. URL: https://docs.python.org/3/library/base64.html.

[34] Python Software Foundation. *csv — CSV File Reading and Writing*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3/library/csv.html.

[35] Python Software Foundation. *datetime — Basic date and time types*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3/library/datetime.html.

[36] Python Software Foundation. *email — An email and MIME handling package*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3/library/email.html.

[37] Python Software Foundation. *General Python FAQ*. Accessed: 2025-10-02. Python Software Foundation. 2025. URL: https://docs.python.org/3/faq/general.html.

[38] Python Software Foundation. *hashlib — Secure hashes and message digests*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3/library/hashlib.html.

[39] Python Software Foundation. *io — Core tools for working with streams*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3/library/io.html.

[40] Python Software Foundation. *json — JSON encoder and decoder*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3/library/json.html.

[41] Python Software Foundation. *mimetypes — Map filenames to MIME types*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3/library/mimetypes.html.

[42] Python Software Foundation. *os — Miscellaneous operating system interfaces*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3/library/os.html.

[43] Python Software Foundation. *pathlib — Object-oriented filesystem paths*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3/library/pathlib.html.

[44] Python Software Foundation. *shutil — High-level file operations*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3/library/shutil.html.

[45] Python Software Foundation. *sys — System-specific parameters and functions*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3.13/library/sys.html.

[46] Python Software Foundation. *time — Time access and conversions*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3.13/library/time.html.

[47] Python Software Foundation. *typing — Support for type hints*. Python Standard Library Documentation, Accessed: 2025-10-03. 2025. URL: https://docs.python.org/3.13/library/typing.html.

[48] Sebastián Ramírez. *FastAPI framework, high performance, easy to learn, fast to code, ready for production*. https://pypi.org/project/fastapi/. Accessed: 2025-10-03. 2025.

[49] The Pandas Development Team. *Powerful data structures for data analysis, time series, and statistics*. https://pypi.org/project/pandas/. Accessed: 2025-10-03. 2025.

[50] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[51] Vue.js Team. *Vue.js - The Progressive JavaScript Framework*. Accessed: 2025-10-04. 2025. URL: https://it.vuejs.org/.

[52] Wikipedia contributors. *Docker (software)*. Accessed: 2025-10-04. 2025. URL: https://en.wikipedia.org/wiki/Docker_(software).

Ringraziamenti

Dedico questo spazio della tesi a tutte le persone che hanno reso possibile la realizzazione di questo lavoro e la conclusione del mio percorso universitario.

Un ringraziamento particolare va al **Prof. Ivan Lanese**, che mi ha seguito con competenza e disponibilità durante la redazione dell'intera tesi, e a **Luca**, CEO di CodeBaker, che mi ha supervisionato durante lo sviluppo del progetto di tesi.

Desidero inoltre ringraziare gli **amici** che hanno fatto parte della mia vita in questi anni e che continueranno a farne parte anche in futuro.

Un ringraziamento speciale va a tutta la **mia famiglia**, che mi ha sostenuto durante l'intera durata di questo percorso accademico. In particolare, ci tengo a ringraziare **mia madre Piera**, per la gentilezza e il costante supporto emotivo durante le fasi più cruciali del mio sviluppo; **mio padre Paolo**, per l'affetto e le risate che ha saputo regalarmi nei momenti di incertezza; e **mia nonna Federica**, per essere stata come una seconda madre.

Infine, ringrazio **me stesso** per la perseveranza dimostrata in questo percorso e mi auguro di trovare la strada giusta per la mia vita.