UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Progettazione di un'architettura software per la gestione indicizzata di Infotecniche con integrazione di un modello LLM per la sintesi automatizzata

Software architecture design for the indexed management of technical documentation integrating an LLM for automated summarization

Relatore:

Emanuele Storti

Tesi di Laurea di:

Seresi Matteo

A.A. 2025/2026

Indice

Introduzione

Negli ultimi anni la trasformazione digitale ha cambiato il modo in cui le organizzazioni archiviano e gestiscono la documentazione. Le aziende producono e consultano un grande quantitativo di manuali, direttive e materiale tecnico. Tuttavia, la semplice trasposizione elettronica genera archivi poco strutturati, rendendo il reperimento delle informazioni necessarie complesso e dispendioso in termini di tempo. Per questo motivo, è fondamentale organizzare tali dati in modo sistematico, così da ridurre i tempi di ricerca e ottimizzare le attività del personale. In questo contesto, l'integrazione di tecniche di Intelligenza Artificiale e di sistemi di automazione rappresenta un'importante innovazione.

Questo elaborato di tesi descrive il lavoro svolto durante il periodo di tirocinio aziendale presso **Elettroquality S.R.L**. L'obiettivo del progetto consiste nella creazione di un'architettura software per la gestione automatizzata di "**Infotecniche**", rendendole fruibili attraverso il gestionale web dell'azienda. Nel caso analizzato, le Infotecniche rappresentano documenti destinati all'assistenza tecnica dei tachigrafi digitali installati sui veicoli commerciali. Si tratta di circolari tecniche e manuali operativi utilizzati per diagnosticare e risolvere le anomalie riscontrate durante il funzionamento dei dispositivi. Questi documenti forniscono ai tecnici indicazioni per: capire specifici codici di errore visualizzati a display o in stampa, chiarire se un messaggio sia dovuto a un guasto interno, a una violazione di sicurezza o a un difetto di comunicazione, e guidare l'operatore nelle procedure d'intervento.

Per rispondere all'esigenza di caricare la documentazione in modo rapido e anche in mobilità (ad esempio da parte di operatori sul campo), il sistema sfrutta un Bot Telegram come punto di ingresso principale. L'utente può inviare al bot file testuali, PDF o semplici immagini presenti nei manuali. Da questo punto, il sistema scarica il file e, mediante algoritmi OCR e librerie dedicate, estrae il testo e le eventuali immagini contenute. Successivamente il contenuto testuale viene elaborato da un modello LLM, incaricato della generazione automatica di un riassunto dell'Infotecnica.

Infine, tutti i dati elaborati vengono archiviati e indicizzati in un database relazionale. Il risultato finale è una pagina web dal quale gli utenti possono consultare le Infotecniche avvalendosi di un motore di ricerca *full-text*, capace di effettuare ricerche contestuali sia sul riassunto generato sia sull'intero contenuto testuale estratto, garantendo il recupero delle informazioni in maniera accurata e pertinente.

La tesi documenta l'intero ciclo di vita del software, dall'analisi del problema alle scelte architetturali, fino ai dettagli implementativi. In particolare, l'elaborato è strutturato nei seguenti capitoli:

- **Nel Capitolo 1** viene descritta l'analisi dei requisiti, definendo gli obiettivi funzionali e non funzionali dell'applicativo.
- **Nel Capitolo 2** vengono presentati gli strumenti e le tecnologie impiegate per lo sviluppo, analizzando il backend, le API esterne e i framework frontend.
- **Nel Capitolo 3** si illustra l'architettura del sistema e la progettazione del database, descrivendo nel dettaglio il flusso di acquisizione asincrona dei dati tramite *webhook* e le tecniche di estrazione del testo grezzo e delle immagini (OCR).
- **Nel Capitolo 4** viene affrontata la fase di implementazione, mostrando le porzioni di codice relative alle strategie di frammentazione dei testi lunghi (*chunking*), all'integrazione del modello IA per la generazione dei riassunti, agli endpoint di ricerca e allo sviluppo dell'interfaccia web.
- Infine, nelle **Conclusioni**, vengono riassunti i traguardi raggiunti dal progetto.

1. Analisi dei requisiti

La fase di analisi dei requisiti ha lo scopo di identificare e descrivere le funzionalità e le caratteristiche che il sistema dovrà implementare al fine di soddisfare le esigenze degli utenti e gli obiettivi del progetto.

1.1 Glossario termini

Termine	Descrizione	Sinonimi
Infotecniche	Circolare, manuale o documento tecnico per fornire istruzioni dettagliate in merito all'uso, alla diagnosi e alla risoluzione delle anomalie dei tachigrafi.	Documento, manuale
LLM	*Large Language Model*: modello linguistico di grandi dimensioni addestrato su vaste quantità di dati testuali, in grado di comprendere e generare linguaggio naturale. Nel progetto, viene impiegato per la sintesi e il riassunto automatico.	-
Bot Telegram	Un'applicazione software automatizzata programmata per interagire con gli utenti all'interno della piattaforma di messaggistica Telegram, ricevendo comandi o file e fornendo risposte.	-
OCR	Tecnologia che permette di riconoscere e trasformare il testo contenuto all'interno di immagini (o PDF scansionati) in formato di testo digitale modificabile e ricercabile.	-
Webhook	Meccanismo attraverso il quale un'applicazione (es. Telegram) invia dati in tempo reale a un'altra applicazione (il server aziendale) non appena si verifica un determinato evento (es. la ricezione di un file).	-
Chunking	Tecnica che consiste nel suddividere un testo lungo in porzioni più piccole e gestibili (*chunk*), mantenendo una sovrapposizione (*overlap*) per non perdere il contesto, al fine di rispettare i limiti di contesto del modello linguistico.	Suddivisione a blocchi

Rasterizzazione	Processo che consiste nella conversione di un'immagine vettoriale o di un documento digitale in un'immagine *raster*, ovvero in un formato di pixel.	-

1.2 Requisiti del progetto

I requisiti costituiscono le fondamenta dell'intero sistema e si suddividono in due categorie:

- **I requisiti funzionali** definiscono i comportamenti e le funzionalità che il software deve obbligatoriamente fornire per soddisfare le esigenze degli utenti.
- **I requisiti non funzionali** definiscono i vincoli di qualità, le performance e i criteri architetturali che il sistema deve rispettare per garantire un servizio affidabile.

1.2.1 Requisiti funzionali

Rappresentiamo l'elenco dei comportamenti e delle funzionalità primarie attese dall'applicativo *(Figura 1.1)*.

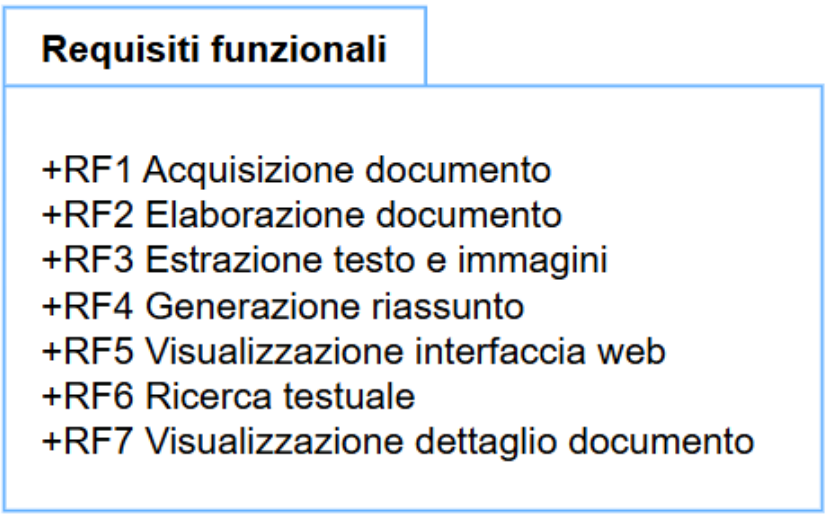


Figura 1.1: Requisiti Funzionali

RF1 - Acquisizione documento

Il sistema deve permettere agli utenti di caricare nuovi documenti tecnici interagendo con un Bot Telegram. Il sistema deve riconoscere automaticamente diverse tipologie di file, tra cui documenti PDF, immagini e file di testo.

RF2 - Elaborazione documento

Al momento della ricezione di un file, il sistema deve prenderlo in carico e avviare l'elaborazione in background, restituendo all'utente un riscontro immediato.

RF3 - Estrazione testo e immagini

Il sistema analizza il file caricato e applicare la strategia di estrazione. Deve essere in grado di estrarre il testo nativo dai PDF e di applicare algoritmi di riconoscimento ottico dei caratteri (OCR) nel caso di immagini o PDF scansionati. Inoltre, deve estrarre e salvare le immagini contenute all'interno dei documenti.

RF4 - Generazione riassunto

Il testo estratto deve essere processato da un modello di Intelligenza Artificiale per generare un riassunto coerente del contenuto.

RF5 - Visualizzazione interfaccia web

Il sistema deve offrire una tabella dei documenti caricati, dotata di paginazione.

RF6 - Ricerca testuale

L'interfaccia web fornisce una barra di ricerca che permette agli utenti di individuare rapidamente i documenti mediante interrogazioni *full-text* sul contenuto e sul riassunto generato.

RF7 - Visualizzazione dettaglio documento

L'utente può selezionare un documento per aprire una vista di dettaglio contenente il riassunto, il testo completo e il link per il download del file originale.

1.2.2 Requisiti non funzionali

Di seguito sono descritti i vincoli prestazionali, di sicurezza e di qualità *(Figura 1.2)*

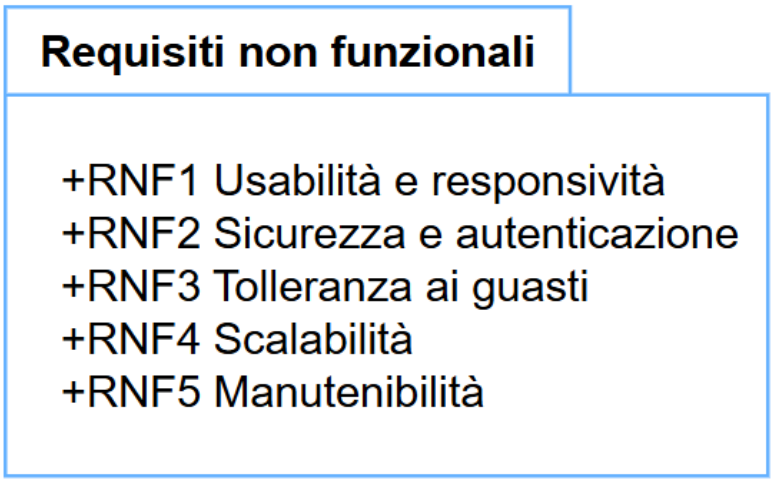


Figura 1.2: Requisiti non funzionali

RNF1 - Usabilità e responsività

L'interfaccia web aziendale deve essere di facile utilizzo (*user-friendly*) e *responsive*, adattandosi alla risoluzione dello schermo.

RNF2 - Sicurezza e autenticazione

Il punto di ingresso dei dati (il *Webhook* di Telegram) deve essere protetto contro accessi non autorizzati. Il sistema deve validare ogni richiesta in entrata verificando la presenza e la correttezza di *token* segreti.

RNF3 - Tolleranza ai guasti

L'architettura deve prevedere meccanismi di *fallback*. Ad esempio, per la generazione del riassunto, se l'interrogazione alle API esterne fallisce il sistema deve implementare una logica di ripetizione automatica temporizzata. Se la generazione del riassunto dovesse comunque fallire, il sistema deve estrarre autonomamente i primi caratteri del documento per garantire la continuità del servizio.

RNF4 - Scalabilità

Il sistema deve garantire un'adeguata separazione delle responsabilità. Nello specifico, i processi leggeri di *routing* web e di ricezione dei *webhook* devono

essere disaccoppiati dai processi più intensivi. Questo deve garantire che il sistema non subisca blocchi dell'interfaccia utente qualora vengano caricati file particolarmente pesanti o elaborati più documenti in contemporanea.

RNF5 – Manutenibilità
Il codice deve essere adeguatamente disaccoppiato in modo da facilitare futuri aggiornamenti, senza impattare sul resto dell'infrastruttura.

1.3 Casi d'Uso

Un diagramma dei casi d'uso è una delle forme principali per descrivere i requisiti di un sistema specificandone il comportamento. Il modello dei casi d'uso *(Figura 1.3)* aiuta la progettazione del sistema dal punto di vista dell'utente finale, andando a facilitare la comunicazione con il sistema.

1.3.1 Diagramma dei casi d'uso

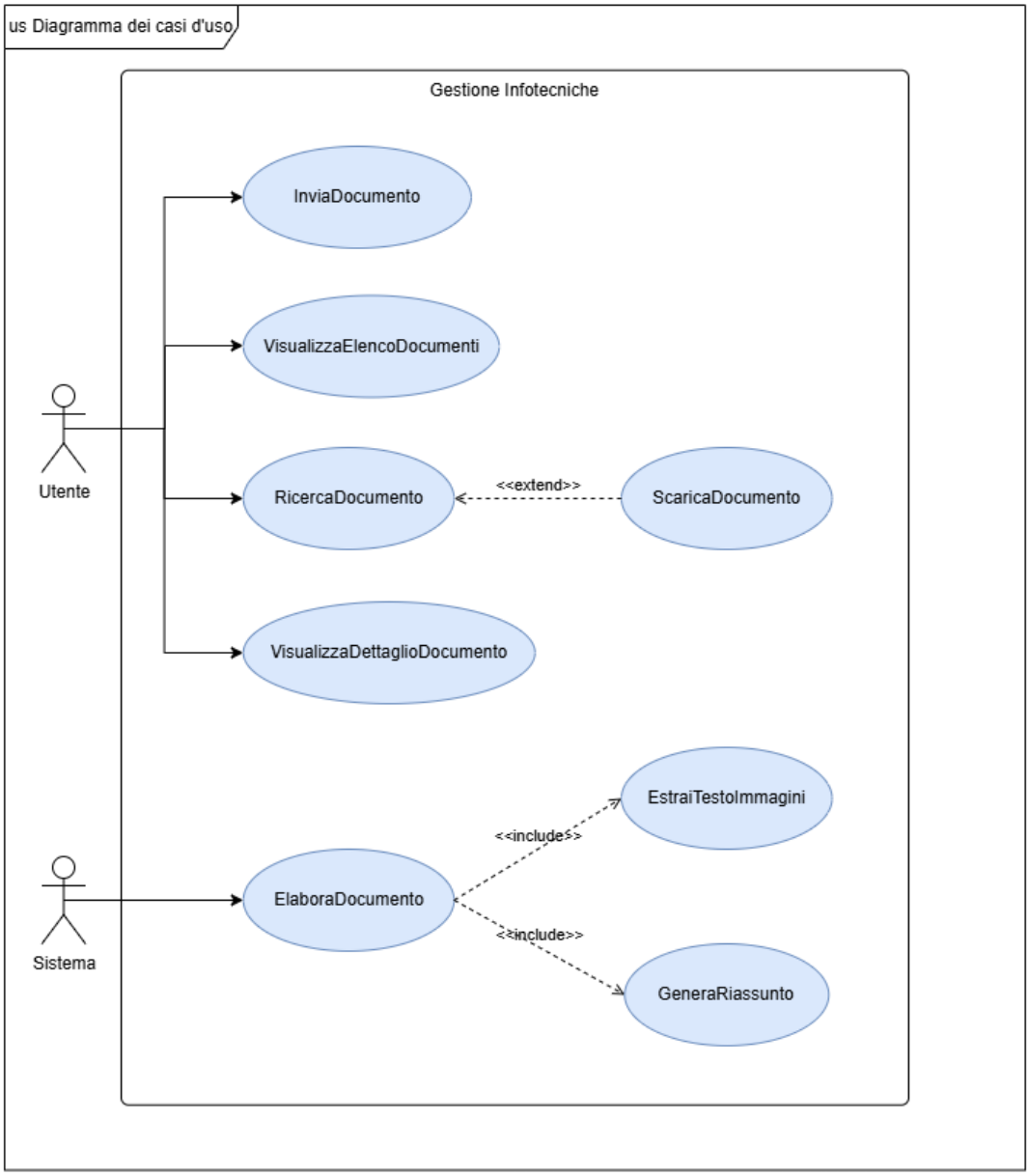


Figura 1.3: Diagramma dei casi d'uso

1.3.2 Descrizione dei casi d'uso

UC1: InviaDocumento	
Descrizione	L'utente invia un'infotecnica al sistema tramite il Bot Telegram per l'indicizzazione
Attore	Utente
Pre-Condizioni	L'utente dispone del documento e dell'accesso al Bot Telegram aziendale
Post-Condizioni	Il documento è salvato nel file system del server, registrato nel database e il processo di elaborazione viene avviato
Sequenza degli eventi principale	1. L'utente invia il file nella chat del Bot 2. Il sistema scarica il file e lo salva sul server 3. Il sistema inserisce il record nel database 4. Il sistema avvia in background il processo di elaborazione 5. Il sistema invia una notifica immediata di ricezione all'utente
Sequenza degli eventi alternativa	○ *Token non valido* ○ *Errore di download*: Il sistema notifica l'utente dell'impossibilità di recuperare il file dai server di Telegram

UC2: VisualizzaElencoDocumenti	
Descrizione	L'utente visualizza la lista delle infotecniche caricate attraverso il portale web
Attore	Utente
Pre-Condizioni	-
Post-Condizioni	-
Sequenza degli eventi principale	1. L'utente accede alla pagina web 2. Il sistema interroga il database per recuperare i record presenti 3. Il sistema mostra i dati in una tabella paginata
Sequenza degli eventi alternativa	-

UC3: RicercaDocumento	
Descrizione	L'utente effettua una ricerca mirata per trovare specifiche infotecniche
Attore	Utente
Pre-Condizioni	-
Post-Condizioni	-
Sequenza degli eventi principale	1. L'utente inserisce una stringa di ricerca nella barra dedicata 2. Il sistema invia la query per l'interrogazione 3. La tabella viene aggiornata dinamicamente con i risultati ritrovati
Sequenza degli eventi alternativa	○ *Nessun risultato:* La tabella viene mostrata vuota con l'avviso che nessun documento corrisponde ai criteri

UC4: VisualizzaDettaglioDocumento	
Descrizione	L'utente consulta i dati completi estratti da un'Infotecnica specifica
Attore	Utente
Pre-Condizioni	Il documento è visibile sul portale web
Post-Condizioni	Apertura di una finestra modale con i dettagli del documento
Sequenza degli eventi principale	1. L'utente clicca sul pulsante "Apri" in corrispondenza del documento desiderato 2. Il sistema recupera dal database il testo completo, il riassunto e l'URL per scaricare il file 3. L'interfaccia mostra il contenuto organizzato all'interno di una finestra modale
Sequenza degli eventi alternativa	○ *Documento non trovato*: Il sistema notifica l'utente che il documento richiesto non è disponibile

UC5: ScaricaDocumento	
Descrizione	L'utente scarica il file originale
Attore	Utente
Pre-Condizioni	La vista di dettaglio del documento è aperta
Post-Condizioni	Trasferimento del file originale sul dispositivo dell'utente
Sequenza degli eventi principale	1. L'utente clicca sul link "Scarica file" presente nel dettaglio 2. Il sistema genera l'URL di download puntando alla directory di storage 3. Il browser avvia il download del file
Sequenza degli eventi alternativa	○ *File mancante*: Se il file è stato rimosso manualmente dal server, il sistema restituisce un errore

UC6: ElaboraDocumento	
Descrizione	Il sistema processa automaticamente il file caricato per estrarne il contenuto
Attore	Sistema
Pre-Condizioni	Un nuovo file è stato caricato
Post-Condizioni	Database aggiornato con testo estratto e riassunto
Sequenza degli eventi principale	1. Il sistema verifica la presenza del file nello storage 2. Il sistema esegue l'estrazione coordinata di testo e immagini 3. Il sistema genera il riassunto tramite modelli linguistici 4. Il sistema aggiorna il record del database con i nuovi dati ottenuti 5. Il sistema invia all'utente il messaggio che il documento è stato elaborato tramite il bot Telegram
Sequenza degli eventi alternativa	-

UC6.1: EstraiTestoImmagini	
Descrizione	Estrazione del testo e delle immagini dal file caricato
Attore	Sistema
Pre-Condizioni	Documento in fase di elaborazione
Post-Condizioni	Creazione di un documento di testo temporaneo e salvataggio delle immagini
Sequenza degli eventi principale	1. Il sistema analizza il tipo di file 2. Se PDF testuale, utilizza strumenti di estrazione layout 3. Se immagine, applica l'OCR 4. Il sistema ricerca ed estrae le immagini incorporate nei documenti
Sequenza degli eventi alternativa	○ *Formato non supportato*: Il sistema termina l'operazione loggando l'impossibilità di procedere

UC6.2: GeneraRiassunto	
Descrizione	Creazione del riassunto tramite un modello LLM
Attore	Sistema
Pre-Condizioni	Testo estratto disponibile
Post-Condizioni	Notifica riassunto generato con successo
Sequenza degli eventi principale	1. Il sistema legge il contenuto testuale estratto 2. Se il testo è eccessivamente lungo, applica una logica di frammentazione 3. Il sistema invia i blocchi di testo al modello IA per la sintetizzazione 4. Il sistema unisce e rifinisce i risultati per produrre un riassunto coerente
Sequenza degli eventi alternativa	○ *Timeout API IA*: Il sistema effettua dei tentativi di riprova o, in caso di fallimento persistente, utilizza i primi paragrafi come riassunto

1.4 Diagramma E-R

Il Modello Entità-Relazione (E-R) è uno strumento concettuale utilizzato nella fase di progettazione dei database per fornire una rappresentazione schematica e ad alto livello dei dati e delle loro interazioni. Gli elementi fondamentali di questo modello sono le *Entità* (che rappresentano classi di oggetti del mondo reale) e gli *Attributi* (che ne descrivono le proprietà).

Analizzando il sistema, è emerso che l'obiettivo primario è l'archiviazione massiva e l'indicizzazione per la ricerca rapida di documenti testuali. Il modello dati è pertanto costituito da una singola entità denominata **INFOTECNICHE** *(Figura 1.4)*, la quale racchiude in sé tutte le informazioni necessarie per identificare, descrivere e ricercare il documento.

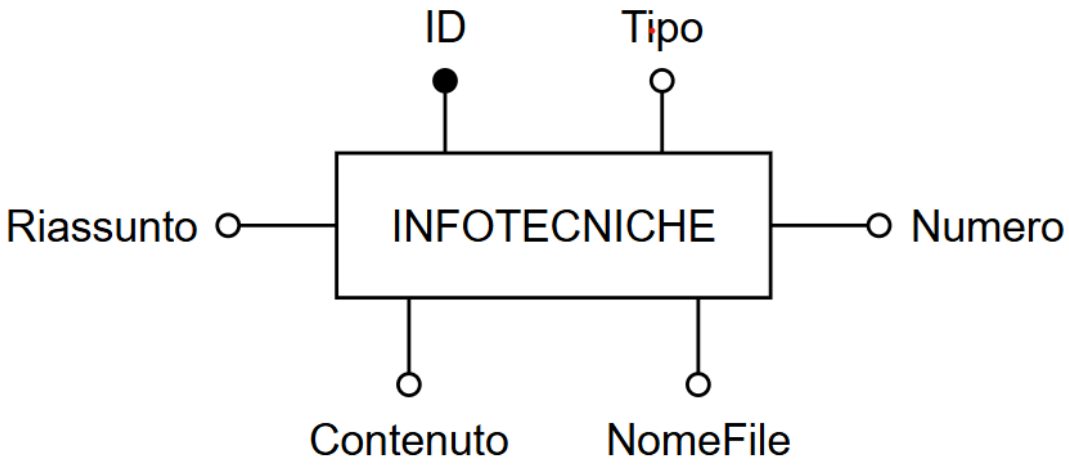


Figura 1.4: Entità INFOTECNICHE

Di seguito si descrivono gli attributi dell'entità:

- **ID (Chiave Primaria):** Identificativo univoco numerico assegnato al documento al momento del caricamento. Utilizzato come riferimento interno dal sistema.
- **NomeFile:** Attributo di tipo stringa che memorizza il nome del file caricato.
- **Tipo:** Attributo di tipo stringa che definisce il formato MIME del file acquisito.

- **Numero:** Attributo di tipo stringa che identifica il "Numero Infotecnica".
- **Contenuto:** Attributo di tipo testo lungo (*LongText*). Contiene l'intero output derivante dall'estrazione del testo.
- **Riassunto:** Attributo di tipo testo lungo (*LongText*). Ospita la sintesi del documento generata dal modello di Intelligenza Artificiale.

2. Tecnologie utilizzate

Questo capitolo mostra le tecnologie adottate per lo sviluppo del sistema informativo. Di seguito vengono analizzati gli ambienti di sviluppo, i linguaggi per il backend e frontend e i sistemi esterni integrati per l'elaborazione dei documenti tramite Intelligenza Artificiale.

2.1 Git e Github

Durante lo sviluppo del progetto è stato necessario adottare un sistema di controllo che consentisse di tracciare le modifiche al codice e ripristinare versioni precedenti in caso di errore. Per questo motivo è stato utilizzato **Git** [1], un sistema di controllo di versione distribuito, open source e gratuito.

Questo strumento permette di registrare ogni singolo aggiornamento del codice tramite i *commit*, garantendo la totale tracciabilità dello sviluppo. Inoltre, la sua architettura basata sui rami (*branching*) consente di sviluppare/testare nuove funzionalità in ambienti isolati, per poi integrarle nel progetto principale (operazioni di *merge*).

Il progetto è stato caricato sulla piattaforma **GitHub** [2]. GitHub è un servizio web di *hosting* che fornisce un'interfaccia per ospitare in *cloud* i *repository* Git. La piattaforma ha consentito di mantenere una copia aggiornata del codice sorgente, riducendo il rischio di perdita dei dati e facilitando il proseguimento dello sviluppo da postazioni differenti.

2.2 Visual Studio Code

L'intero ciclo di stesura del codice sorgente è stato condotto utilizzando **Visual Studio Code** [3], un code editor avanzato e multipiattaforma sviluppato da Microsoft.

La scelta di questo strumento è stata dettata l'utilizzo di diversi linguaggi di programmazione. All'interno dello stesso ambiente è stato infatti possibile sviluppare i componenti PHP dedicati alla gestione dei *webhook* e delle API, gli script Python per l'elaborazione dei documenti, oltre alle pagine HTML, ai fogli di stile CSS, al codice JavaScript e alle query SQL impiegate dal gestionale web. L'utilizzo di un unico ambiente di sviluppo ha semplificato l'organizzazione del progetto, consentendo di gestire in modo uniforme tutti i componenti software.

2.3 Putty

Per la configurazione e l'amministrazione del server di produzione è stato utilizzato **PuTTY** [4], un client *open source* utilizzato per le connessioni di rete tramite protocollo SSH (*Secure Shell*).

Poiché il progetto è stato distribuito su un server Linux remoto, si è reso necessario disporre di un accesso diretto al terminale per eseguire le operazioni di configurazione e manutenzione dell'ambiente di esecuzione.

Nel rispetto delle policy di sicurezza aziendali, le attività sono state svolte in affiancamento al tutor aziendale. Tramite l'interfaccia a riga di comando è stato possibile eseguire diverse operazioni fondamentali per il corretto funzionamento dell'applicativo, tra cui:

- Creazione di un ambiente virtuale (`virtualenv`) per isolare le dipendenze Python del progetto e installazione dei pacchetti necessari tramite il file `requirements.txt`.
- Configurazione del file `.env`, utilizzato per memorizzare in modo sicuro le credenziali del database e le chiavi API, evitando la loro esposizione nel codice sorgente.
- Assegnazione dei corretti privilegi di lettura, scrittura ed esecuzione ai file e alle directory.
- Registrazione e verifica del *Webhook* di Telegram, associando l'indirizzo HTTPS del server al Bot per abilitare la ricezione automatica dei file.

2.4 FileZilla

Per gestire il trasferimento del codice sorgente dall'ambiente di sviluppo locale al server, è stato impiegato **FileZilla** [5]. Sebbene FileZilla nasca principalmente per gestire connessioni FTP tradizionali, all'interno di questo progetto è stato configurato per operare esclusivamente tramite **SFTP** (*SSH File Transfer Protocol*).

L'impiego di FileZilla si è rivelato particolarmente utile grazie alla sua interfaccia grafica intuitiva, che ha permesso il trasferimento rapido degli script PHP, Python e dei file di interfaccia (HTML/CSS/JS) verso la *root* del server web Nginx [6].

2.5 Backend

Il backend costituisce il livello dell'applicativo responsabile dell'elaborazione dei dati, della comunicazione con il database e dell'integrazione con i servizi esterni.
Al fine di soddisfare i requisiti definiti in fase di analisi, si è scelto di adottare un'architettura ibrida basata sull'impiego di due tecnologie distinte: una utilizzata per la gestione delle richieste web e delle API, l'altra dedicata alle operazioni più pesanti come l'elaborazione dei documenti.

2.5.1 PHP

PHP (Hypertext Preprocessor) [7] è un linguaggio di scripting *server-side* ideato specificamente per lo sviluppo web.

Nel progetto è stato impiegato per realizzare il backend del gestionale, occupandosi della ricezione dei *webhook* provenienti da Telegram, dell'esposizione delle API utilizzate dal frontend e della comunicazione con il database MySQL. La scelta di PHP è stata motivata dalla sua integrazione con il server web Nginx e dalla presenza di un'infrastruttura aziendale già basata su questa tecnologia, che ne ha facilitato l'adozione e la distribuzione. Il suo ruolo consiste quindi nel validare le richieste ricevute, gestire gli aspetti di sicurezza e coordinare l'interazione tra il client, il database e i processi di elaborazione eseguiti in Python.

2.5.2 Python

Python [8] è un linguaggio di programmazione di alto livello, interpretato e multiparadigma.

Python è stato impiegato per gestire le operazioni di elaborazione dei documenti, sfruttando librerie specifiche per l'estrazione del testo dai PDF e il riconoscimento ottico dei caratteri (OCR). Pur essendo un linguaggio interpretato, Python si è dimostrato adeguato per l'esecuzione dei processi asincroni in background.

2.6 Elaborazione documento

Poiché le Infotecniche possono essere fornite dagli operatori in formati differenti, il sistema deve essere in grado di estrapolare le informazioni in modo affidabile a prescindere dal tipo di documento.

Sono stati adottati due strumenti: il primo dedicato all'estrazione del testo dai PDF, il secondo basato sul riconoscimento ottico dei caratteri per l'analisi di immagini e documenti scansionati.

2.6.1 Poppler-utils

La gestione nativa dei file in formato PDF è stata affidata a **Poppler-utils** [9], una solida suite di strumenti software *open source* basata sulla libreria di rendering Poppler.

Essendo il PDF lo standard per la distribuzione di documenti tecnici, è stato necessario l'adozione questa tecnologia. Da un lato, permette l'estrazione del testo vettoriale nativo preservando la struttura spaziale e il *layout* originale; dall'altro lato, offre strumenti di conversione per trasformare le pagine PDF in immagini ad alta risoluzione o per estrapolare fotografie incorporate.

2.6.2 Tesseract OCR

Per gestire documenti privi di testo digitale (come fotografie scattate sul campo o PDF derivanti da scansioni cartacee), si è reso necessario l'utilizzo di un motore di riconoscimento ottico dei caratteri (OCR). La scelta è ricaduta su

Tesseract OCR [10], un software *open source* originariamente sviluppato da Hewlett-Packard e oggi manutenuto da Google.

Tesseract è stato selezionato per la sua capacità di riconoscere pattern alfanumerici all'interno di immagini, supportando un'ampia varietà di linguaggi. La sua natura eseguibile da terminale lo rende integrabile in flussi di lavoro automatizzati e asincroni, garantendo che nessun documento venga ignorato o scartato a causa della semplice assenza di un livello testuale vettoriale di base.

2.7 API esterne

Per alcune funzionalità del sistema è stato necessario integrare servizi esterni tramite **API** (*Application Programming Interface*). Questo approccio ha permesso di delegare specifiche operazioni a piattaforme specializzate, evitando di implementare internamente componenti già disponibili.

Nello specifico, il progetto si appoggia a due servizi esterni con scopi diversi: uno orientato alla comunicazione e alla gestione dell'interfaccia utente in mobilità, e l'altro dedicato all'elaborazione avanzata del linguaggio naturale (NLP) delegata in ambiente cloud.

2.7.1 Telegram Bot API

Per fornire agli utenti un punto di accesso al sistema che fosse immediato e utilizzabile in mobilità, si è scelto di utilizzare **Telegram Bot API** [11]. Questa tecnologia permette di integrare un bot Telegram con il backend aziendale, consentendo la ricezione automatica dei file inviati dagli utenti.

La scelta di utilizzare Telegram ha permesso di evitare lo sviluppo di un'applicazione mobile dedicata, sfruttando invece una piattaforma già disponibile e conosciuta. Attraverso il meccanismo dei *webhook*, il server riceve una notifica immediata quando viene caricato un nuovo documento e può avviare il processo di elaborazione. L'integrazione con le API di Telegram viene quindi utilizzata come punto di ingresso del sistema gestendo l'acquisizione dei documenti e il relativo stato di elaborazione.

2.7.2 Hugging Face Inference API

Per la generazione dei riassunti dai testi estratti, si è scelto di integrare logiche di *Natural Language Processing* (NLP).

Il progetto si è affidato alle **Inference API** fornite da **Hugging Face** [12], la principale piattaforma *cloud* per il *Machine Learning open source*. Queste API permettono di interrogare modelli di Intelligenza Artificiale residenti su server remoti. Questa scelta è motivata dal fatto che l'esecuzione in locale di modelli linguistici di grandi dimensioni richiede in particolare GPU ad alte prestazioni e con l'utilizzo delle Inference API ha quindi consentito di demandare l'elaborazione ai server di Hugging Face.

2.8 MySQL

Il livello di persistenza dei dati è stato implementato utilizzando **MySQL** [13], principalmente per sfruttare il suo motore nativo di *Full-Text Search*. Questa tecnologia di indicizzazione avanzata è fondamentale per le ricerche testuali complesse all'interno di grandi moli di documenti testuali, garantendo tempi di risposta rapidi e restituendo i risultati in ordine di pertinenza.

Per l'amministrazione del database sul server è stato impiegato **Adminer** [14]. Adminer è un'interfaccia web di gestione ed è ideale per la gestione rapida delle tabelle e il collaudo delle query durante le configurazioni sul server.

2.9 Frontend

Il frontend ha l'obiettivo di fornire agli utenti un'interfaccia semplice per consultare e ricercare le Infotecniche. Il suo sviluppo si basa sui principali standard del web, organizzati secondo una separazione tra la struttura, la logica applicativa e la visualizzazione avanzata dei dati.

2.9.1 HTML e CSS

L'utilizzo di **HTML** [15] ha garantito la creazione di una struttura solida, accessibile e ottimizzata per i browser moderni. Il **CSS** [16] è stato impiegato per definire l'intera veste grafica dell'applicativo. In particolare, poiché il sistema è destinato all'uso anche tramite, l'interfaccia è stata progettata seguendo i principi *responsive*, la struttura è in grado di adattarsi ad ogni tipo qualsiasi schermo garantendo la leggibilità.

2.9.2 JavaScript e jQuery

Il comportamento dinamico del client è stato interamente affidato a **JavaScript** [17], supportato dall'adozione della libreria **jQuery** [18]. La scelta di integrare jQuery è stata motivata dalla sua affidabilità nel semplificare l'interazione con il *Document Object Model* (DOM) e per la gestione standardizzata e *cross-browser* delle chiamate asincrone. Grazie all'implementazione AJAX (*Asynchronous JavaScript and XML*), il frontend

è in grado di inviare dati e ricevere risposte in formato JSON [19] dal server in background. Questo approccio ha permesso di trasformare l'interfaccia in una *Single Page Application* (SPA) semplificata: la pagina non necessita mai di essere ricaricata interamente, restituendo all'utente un'esperienza di navigazione fluida e istantanea.

2.9.3 DataTables

Per la visualizzazione e l'esplorazione dell'archivio documentale, si è scelto di integrare **DataTables** [20], un *plugin* per jQuery. DataTables è stato impiegato sfruttando la sua modalità di *Server-Side Processing*. Invece di caricare e renderizzare l'intero database sul browser dell'utente, questa tecnologia permette di delegare l'elaborazione della paginazione, dell'ordinamento delle colonne e del filtraggio delle ricerche, direttamente al server e a MySQL. In questo modo, l'interfaccia richiede e renderizza solo i record strettamente necessari alla visualizzazione corrente a prescindere dal numero di Infotecniche archiviate.

3. Progettazione

3.1 Infrastruttura di Sistema

Il sistema gestionale delle Infotecniche è progettato per integrarsi all'interno dell'infrastruttura IT aziendale.

Il rilascio dell'applicativo è stato effettuato su una macchina virtuale ospitata presso l'infrastruttura *cloud* di Aruba. Il server di produzione utilizza **AlmaLinux** [21], un sistema operativo basato su ambiente Linux e pensato per garantire stabilità e affidabilità nella gestione di servizi applicativi.

L'esposizione del sistema sulla rete Internet è gestita tramite **Nginx**, che svolge il ruolo di punto di accesso per le comunicazioni esterne. Nginx è configurato per intercettare il traffico in entrata e gestire simultaneamente due flussi di rete:

- **Traffico API (*webhook*):** ricezione delle richieste HTTP provenienti dai server di Telegram contenenti i dati relativi ai documenti inviati dagli utenti, con successivo inoltro agli script PHP incaricati della gestione.
- **Traffico Utente (frontend):** distribuzione delle risorse web dell'applicativo (HTML, CSS e JavaScript) e gestione delle richieste generate dagli utenti tramite interfaccia frontend.

Dal punto di vista logico, il server rappresenta il punto di integrazione tra tre componenti principali del sistema:

- **Client mobile (Telegram Bot):** utilizzato dagli operatori come strumento di caricamento dei documenti e ricezione delle notifiche relative all'elaborazione.
- **Server aziendale (AlmaLinux):** responsabile della gestione dell'applicativo, del salvataggio dei file, dell'estrazione testuale e dell'interazione con il database MySQL.

- **Servizio cloud esterno (Hugging Face):** utilizzato esclusivamente per l'elaborazione dei modelli di Intelligenza Artificiale dedicati alla generazione dei riassunti dei documenti.

3.2 Database

Il database rappresenta l'archivio digitale responsabile della memorizzazione delle informazioni estratte dai documenti. La progettazione della struttura dati ha richiesto la traduzione del modello concettuale definito in fase di analisi in un modello implementato tramite MySQL

3.2.1 Sicurezza

Per proteggere il database da accessi non autorizzati l'interfaccia di amministrazione web Adminer è disponibile esclusivamente all'interno della rete privata aziendale. Pertanto, l'accesso ad Adminer è consentito tramite l'indirizzo IP interno della rete privata (172...*/adminer). L'accesso a questa rete viene gestito da **OpenVPN** [22], una soluzione open source per la realizzazione di connessioni VPN sicure.

3.2.2 Traduzione del modello e-r

Nella fase di analisi dei requisiti è stato sviluppato il modello concettuale Entità-Relazione (E-R). Questo modello logico è stato successivamente tradotto nel modello fisico implementato all'interno del DBMS MySQL.

In primo luogo, si è proceduto alla creazione del database relazionale, denominato **MyTACHO_infotecniche**. Al suo interno, l'entità principale del sistema è stata trasposta nell'unica tabella denominata **Info_Tecniche**, definendo le rispettive colonne e i tipi di dato *(Figura 3.1)*.

Colonna	Tipo
ID	int(11) *Auto incremento*
Tipo	varchar(255) *NULL*
Numero	varchar(255) *NULL*
Nome_File	varchar(255) *NULL*
Contenuto	text *NULL*
Riassunto	text *NULL*

Figura 3.1: Struttura tabella Info_Tecniche

Inoltre, viene definito l'indice di tipo **FULLTEXT** applicato alla colonna Contenuto *(Figura 3.2)*. Questa scelta permette di effettuare ricerche testuali più efficienti rispetto all'utilizzo dell'operatore LIKE.

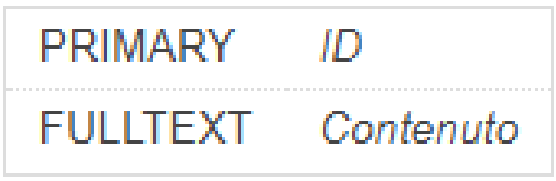


Figura 3.2: Indice FULLTEXT

L'ambiente di produzione del database viene gestito tramite Adminer *(Figura 3.3)*, che consente di visualizzare la struttura delle tabelle e monitorare i record progressivamente inseriti nel sistema *(Figura 3.4)*.

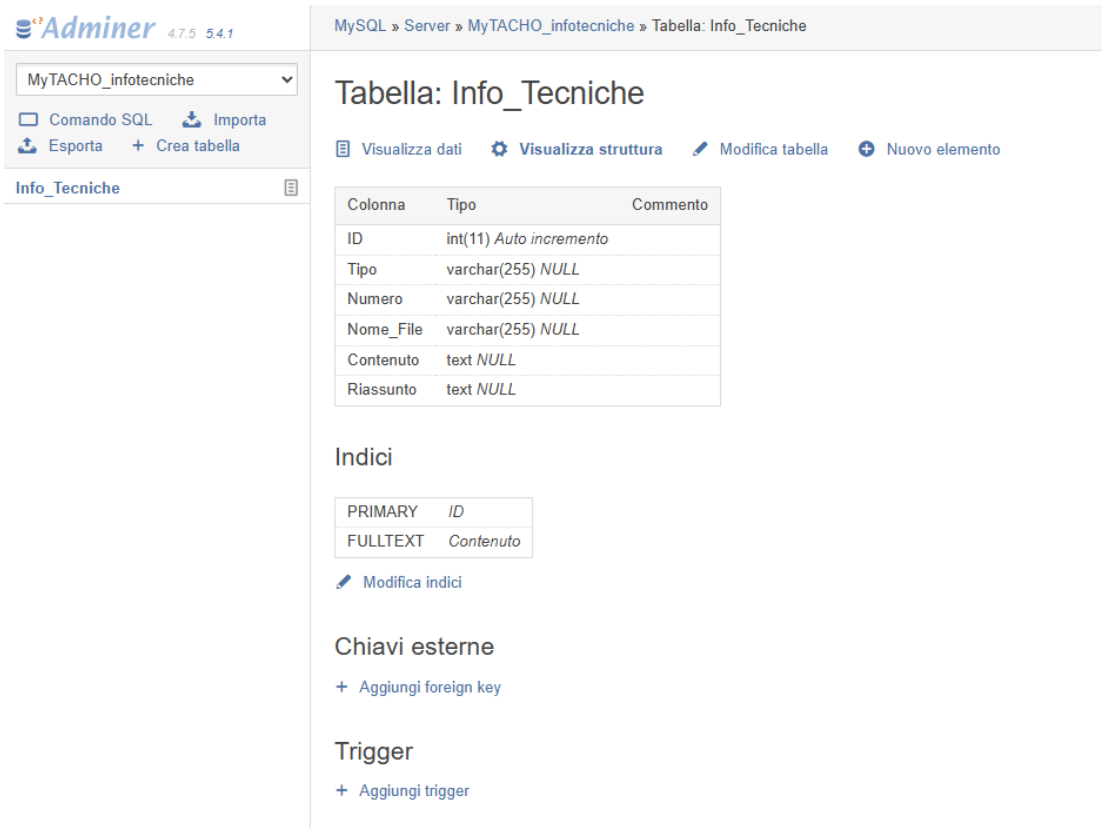


Figura 3.3: Ambiente di gestione Adminer

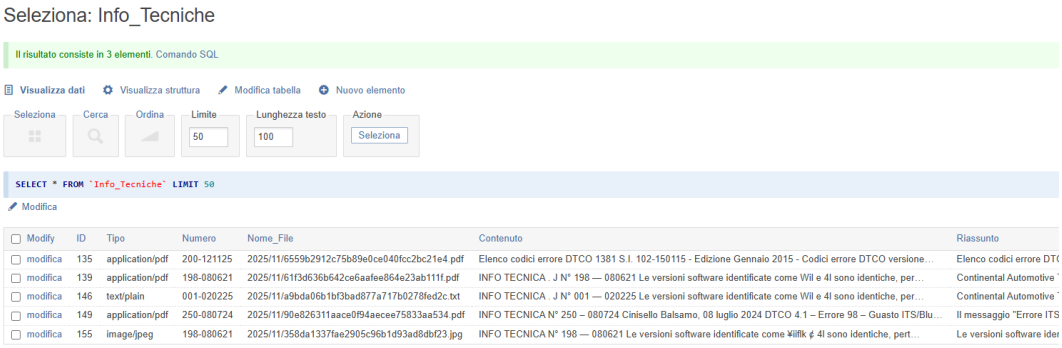


Figura 3.4: Record popolati all'interno della tabella

3.3 Data flow

In questa sezione viene descritto il flusso di elaborazione asincrono dell'Infotecnica, dall'invio del documento fino alla sua archiviazione nel database. Il processo è suddiviso in quattro fasi:

- **Acquisizione documento**
- **Disaccoppiamento asincrono**
- **Estrazione e riassunto**
- **Consolidamento**

3.3.1 Acquisizione documento

Il processo ha inizio quando un utente invia un file all'interno della chat del Bot Telegram. L'infrastruttura di Telegram intercetta il messaggio e genera una richiesta HTTP POST indirizzata al server aziendale. Questa richiesta veicola un *payload* in formato JSON contenente le informazioni necessarie per identificare l'utente e recuperare il documento. Il modulo di ricezione convalida i parametri del pacchetto e interroga nuovamente Telegram Bot API per scaricare fisicamente il documento. Il file viene salvato all'interno del server, organizzato per anno/mese e viene inviato un messaggio di notifica all'utente per confermare l'avvenuta ricezione e la presa in carico dell'operazione.

3.3.2 Disaccoppiamento asincrono

La fase di elaborazione del documento può richiedere diverso tempo a causa delle operazioni di estrazione del testo e della chiamata del modello di Intelligenza Artificiale. Ciò provocherebbe un errore di timeout sui server di Telegram, i quali interpreterebbero il ritardo come un malfunzionamento del sistema, bloccando ulteriori interazioni.

Per ovviare a questa limitazione, viene delegata l'elaborazione avviando il processo in background, svincolandolo dal ciclo di vita della richiesta web. Successivamente viene chiusa la connessione rendendo il bot pronto ad accogliere nuovi documenti.

3.3.3 Estrazione e riassunto

Una volta avviato il processo in background, il sistema analizza il tipo di file ricevuto per scegliere la corretta modalità di estrazione del contenuto:

- Se si tratta di un PDF vettoriale, invoca le librerie della suite *Poppler-utils* per estrarre la componente testuale.
- Se il file è un'immagine o un PDF derivante da una scansione, viene utilizzato *Tesseract OCR* per il riconoscimento ottico dei caratteri.

Una volta ottenuto il testo grezzo, il processo suddivide il contenuto in frammenti più piccoli in modo da rispettare i limiti imposti dal modello linguistico e trasmette i dati alle *Inference API* di Hugging Face, mettendosi in attesa della restituzione del riassunto.

3.3.4 Consolidamento

Ricevuto il riassunto, tutti i dati vengono memorizzati all'interno del database relazionale e l'Infotecnica è visibile e consultabile su interfaccia web. Tramite questa pagina, gli utenti possono esplorare l'intero archivio, leggere i riassunti generati ed effettuare ricerche testuali sui documenti.

Al termine dell'elaborazione viene inviato un messaggio di successo all'utente, confermando l'avvenuta archiviazione e fornendo un'anteprima del riassunto generato.

3.3.5 Diagramma di sequenza

Il flusso descritto è rappresentato nel diagramma di sequenza *(Figura 3.5)*, che illustra il passaggio asincrono dei messaggi tra gli attori e i moduli di sistema.

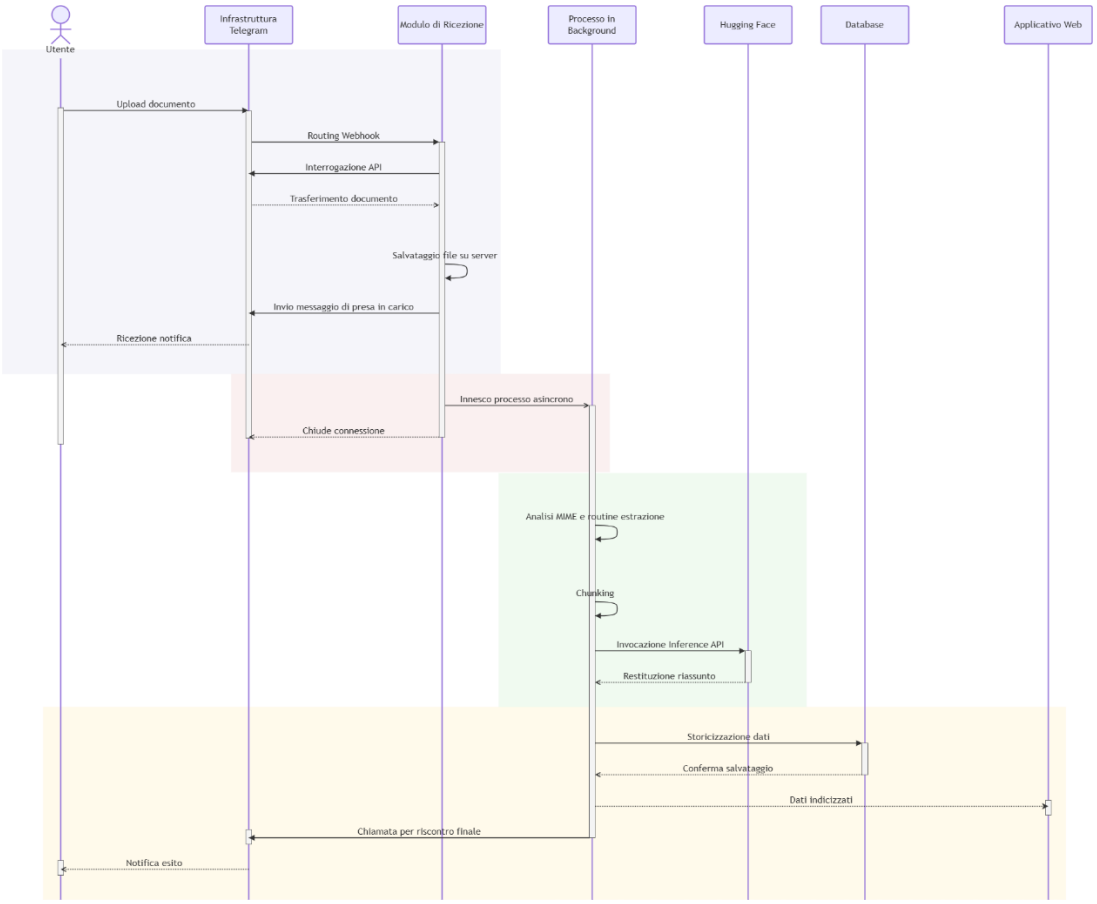


Figura 3.5: Diagramma di sequenza

3.4 Progettazione Back-end

L'architettura backend è stata progettata secondo una suddivisione a livelli, ispirata al pattern MVC (*Model-View-Controller*) [23].

Vengono separate le responsabilità permettendo di isolare le diverse funzionalità dell'applicativo: il livello esterno gestisce le comunicazioni con i client, il livello intermedio contiene la logica applicativa e il livello più basso si occupa della gestione dei dati (*Figura 3.6*).

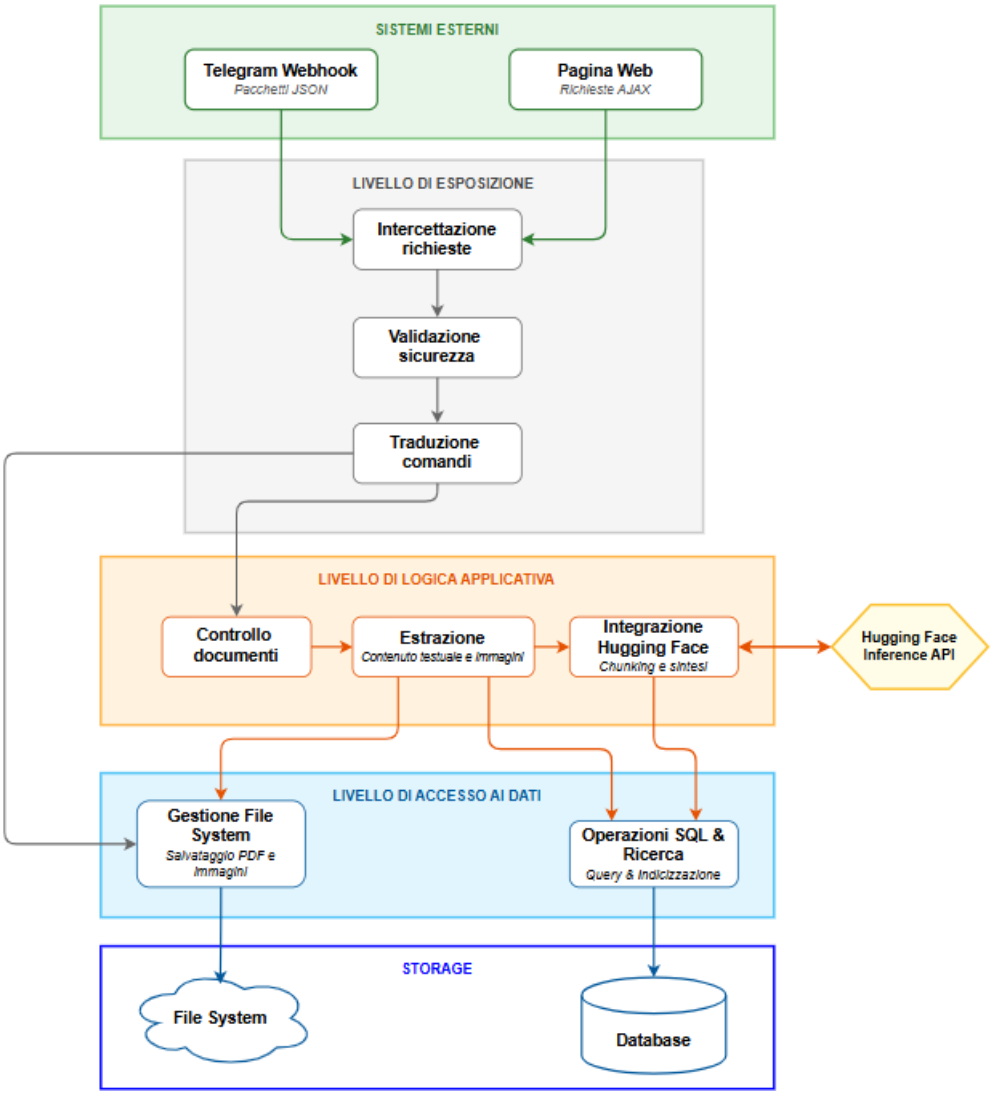


Figura 3.6: Architettura backend

3.4.1 Livello di esposizione

Questo livello rappresenta il punto di ingresso del backend e gestisce le comunicazioni provenienti dai sistemi esterni.

Ha il compito di intercettazione e validazione: riceve i pacchetti JSON dal *Webhook* di Telegram o le richieste asincrone (AJAX) provenienti dalla pagina web, ne verifica l'autenticità tramite i token di sicurezza e traduce queste richieste in comandi comprensibili per il sistema interno. Una volta ricevuto l'input, lo inoltra al livello sottostante.

3.4.2 Livello di logica applicativa

Il livello applicativo contiene le principali operazioni che regolano il funzionamento del gestionale. Esso agisce da intermediario, mantiene separata la gestione dei dati dalla logica applicativa, in questo modo il livello di esposizione rimane indipendente dalle modalità con cui i dati vengono gestiti.

Al suo interno vengono gestite le procedure di elaborazione dei documenti, tra cui l'analisi del formato ricevuto e la scelta dello strumento di estrazione più adatto. Inoltre, questo livello gestisce l'integrazione con il servizio esterno di Hugging Face Inference API, suddividendo i testi troppo lunghi in porzioni compatibili con il modello linguistico e ottenendo un unico riassunto.

3.4.3 Livello di accesso ai dati

Il livello di accesso ai dati è incaricato esclusivamente della persistenza, all'indicizzazione e al recupero fisico delle informazioni.

Gestisce il salvataggio dei file nel *file system* e le operazioni verso il database. I dati ricevuti dal livello applicativo vengono trasformati in operazioni SQL per l'inserimento, la modifica e il recupero delle informazioni. Inoltre, viene gestita la ricerca testuale, utilizzata per individuare i documenti più pertinenti in base al contenuto estratto. La separazione tra logica applicativa e accesso ai dati permette di mantenere il database indipendente dal resto dell'applicativo

3.5 Progettazione Front-end

Il frontend costituisce il punto di accesso utilizzato dagli operatori per consultare e ricercare le Infotecniche archiviate nel sistema.

L'architettura segue un approccio basato su una *Single Page Application* semplificata: la struttura principale della pagina viene caricata inizialmente dal browser, mentre le successive operazioni vengono gestite tramite richieste asincrone al server. Le azioni dell'utente, come la ricerca o la selezione di un documento, non richiedono il caricamento della pagina. I dati vengono recuperati tramite chiamate AJAX e aggiornati dinamicamente solo nelle sezioni interessate dell'interfaccia.

L'interfaccia presenta due viste:

- **Vista della tabella:** rappresenta la schermata principale e contiene l'elenco delle Infotecniche archiviate. Attraverso questa sezione l'utente può effettuare ricerche e selezionare i documenti specifici.
- **Vista di dettaglio:** permette di visualizzare le informazioni relative al documento selezionato. L'utente può consultare il riassunto generato, visualizzare il contenuto e scaricare il file originale.

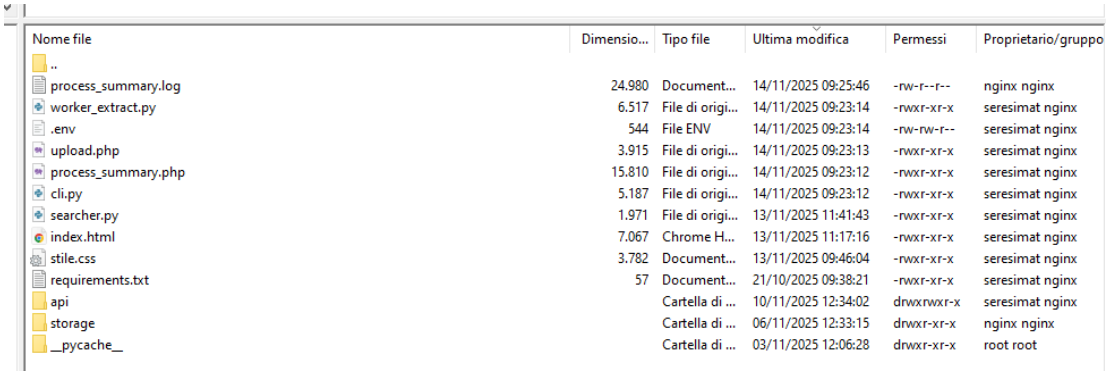
l flusso di navigazione per la ricerca testuale prevede l'inserimento di una o più parole chiave nell'apposito campo di ricerca, alla quale segue una richiesta asincrona verso il backend. Il server restituisce i risultati ordinati in base alla rilevanza della ricerca e l'interfaccia aggiorna la tabella visualizzata.

4. Implementazione

In questo capitolo viene descritta l'implementazione del sistema, analizzando la struttura del progetto e il funzionamento dei principali file sorgente che compongono il sistema.

4.1 Struttura del progetto

Prima dell'analisi del codice sorgente, è necessario mostrare l'ambiente di sviluppo e la struttura organizzativa dei file *(Figura 4.1)*.



Nome file	Dimensio...	Tipo file	Ultima modifica	Permessi	Proprietario/gruppo
..					
process_summary.log	24.980	Document...	14/11/2025 09:25:46	-rw-r--r--	nginx nginx
worker_extract.py	6.517	File di origi...	14/11/2025 09:23:14	-rwxr-xr-x	seresimat nginx
.env	544	File ENV	14/11/2025 09:23:14	-rw-rw-r--	seresimat nginx
upload.php	3.915	File di origi...	14/11/2025 09:23:13	-rwxr-xr-x	seresimat nginx
process_summary.php	15.810	File di origi...	14/11/2025 09:23:12	-rwxr-xr-x	seresimat nginx
cli.py	5.187	File di origi...	14/11/2025 09:23:12	-rwxr-xr-x	seresimat nginx
searcher.py	1.971	File di origi...	13/11/2025 11:41:43	-rwxr-xr-x	seresimat nginx
index.html	7.067	Chrome H...	13/11/2025 11:17:16	-rwxr-xr-x	seresimat nginx
stile.css	3.782	Document...	13/11/2025 09:46:04	-rwxr-xr-x	seresimat nginx
requirements.txt	57	Document...	21/10/2025 09:38:21	-rwxr-xr-x	seresimat nginx
api		Cartella di ...	10/11/2025 12:34:02	drwxrwxr-x	seresimat nginx
storage		Cartella di ...	06/11/2025 12:33:15	drwxr-xr-x	nginx nginx
__pycache__		Cartella di ...	03/11/2025 12:06:28	drwxr-xr-x	root root

Figura 4.1: Struttura del progetto su FileZilla

Il progetto è suddiviso in:

- **Radice del progetto:** costituisce lo spazio principale. Contiene i file per il frontend `index.html` e `stile.css` e i moduli principali del backend: `process_summary.php`, `worker_extract.py`, `cli.py` e `searcher.py`. In quest'area risiedono anche i file di log e le configurazioni di ambiente.
- **La directory api/:** raggruppa gli script PHP dedicati alla comunicazione client-server tramite payload in formato JSON . Include gli endpoint invocati dall'interfaccia web (`datatables_search.php` e `view_doc.php`) e lo script che funge da *webhook* (`telegram_webhook.php`).
- **La directory storage/:** rappresenta l'area di *file system* dedicata alla all'archiviazione dei documenti. I file scaricati vengono organizzati in sottocartelle per mese e anno.

4.1.1 Sicurezza dell'ambiente

Per garantire un livello di sicurezza adeguato le variabili d'ambiente sono gestite tramite un file `.env`, prevenendo l'inserimento dei dati sensibili direttamente nel codice.

All'interno del file sono definiti:

- Le credenziali per la connessione al database MySQL (`DB_HOST`, `DB_NAME`, `DB_USER`, `DB_PASS`).
- I percorsi assoluti degli eseguibili di sistema (`PDFTOTEXT_BIN`, `TESSERACT_BIN`, ecc.).
- I token di autenticazione verso le API (`TELEGRAM_BOT_TOKEN` e l'`HF_TOKEN`)

Infine, le dipendenze Python sono state dichiarate all'interno del file `requirements.txt`, assicurando che il server disponga sempre delle librerie necessarie.

4.2 Webhook endpoint

Il file `api/telegram_webhook.php` gestisce la ricezione dei messaggi provenienti da Telegram e avvia il processo di acquisizione dei documenti.

4.2.1 Sicurezza e autenticazione

La prima fase dello script verifica la presenza del token di autenticazione ricevuto tramite header HTTP *(Figura 4.2)*.

```php
// === CHECK SECRET TOKEN ===
$headers = getallheaders();
if (empty($headers['X-Telegram-Bot-Api-Secret-Token']) ||
    trim($headers['X-Telegram-Bot-Api-Secret-Token']) !== $WEBHOOK_SECRET) {
    http_response_code(403);
    echo json_encode(['ok' => false, 'error' => 'Invalid secret token']);
    exit;
}
```

Figura 4. 2: Controllo header di sicurezza

Il codice PHP estrae questo valore e lo confronta con `$WEBHOOK_SECRET`. Se non coincide, la connessione viene interrotta (HTTP 403 *Forbidden*).

4.2.2 Interpretazione del messaggio

Superato il blocco di sicurezza, il *payload* JSON in ingresso viene decodificato per distinguere il tipo di contenuto. Il codice analizza la struttura dell'array `$update['message']`:

- se individua l'oggetto document, estrae l'identificativo univoco (`file_id`) *(Figura 4.3)*;

```php
if (isset($update['message'])) {
    $msg = $update['message'];
    $chat_id = $msg['chat']['id'] ?? null;

    // Document (preferito) oppure photo (larghezza maggiore)
    if (!empty($msg['document'])) {
        $file_id = $msg['document']['file_id'];
        $orig_filename = $msg['document']['file_name'] ?? ('doc_' . time() . '.pdf');
```

Figura 4.3: Oggetto document

- se è una foto (oggetto `photo`), il sistema seleziona l'ultima occorrenza dell'array, che corrisponde all'immagine con la risoluzione più elevata *(Figura 4.4)*;

```
} elseif (!empty($msg['photo'])) {
    // prendi l'ultima (più grande)
    $photos = $msg['photo'];
    $last = end($photos);
    $file_id = $last['file_id'];
    $orig_filename = 'photo_' . $file_id . '.jpg';
```

Figura 4.4: Oggetto photo

- se l'input è un testo, il sistema invia un messaggio che invita l'utente a caricare un documento valido *(Figura 4.5)*.

```
} elseif (!empty($msg['text'])) {
    send_telegram($BOT_TOKEN, 'sendMessage', [
        'chat_id' => $msg['chat']['id'],
        'text' => "Per indicizzare l'infotecnica, invia un documento, una foto o un file di testo."
    ]);
    http_response_code(200);
    echo json_encode(['ok'=>true,'note'=>'no file']);
    exit;
}
```

Figura 4.5: Oggetto text

4.2.3 Download Sicuro, Hashing e Identificazione

Ottenuto il `file_id`, lo script ricava il percorso fisico del file sui server di Telegram e gestisce il download tramite la libreria **`CURL`** [24], scrivendo direttamente su file per evitare caricamenti completi in memoria.

Una volta acquisito, il file viene analizzato tramite la funzione `finfo_file()`: il sistema determina il reale tipo MIME (es. application/pdf), ignorando l'estensione nominale per prevenire tentativi di *spoofing (Figura 4.6)*.

```php
// --- Rilevo MIME reale del file appena salvato
$detected_mime = 'application/octet-stream';
if (function_exists('finfo_open')) {
    $finfo = finfo_open(FILEINFO_MIME_TYPE);
    if ($finfo !== false) {
        $m = finfo_file($finfo, $target_path);
        if ($m !== false && trim($m) !== '') $detected_mime = $m;
        finfo_close($finfo);
    }
} elseif (function_exists('mime_content_type')) {
    $m = mime_content_type($target_path);
    if ($m !== false && trim($m) !== '') $detected_mime = $m;
}
```

Figura 4.6: Determinazione tipo MIME

Infine, se il tipo MIME rilevato indica che il file è un documento di testo semplice, il sistema evita di delegare il lavoro al motore di estrazione Python. Apre direttamente il file in PHP, normalizza la formattazione (convertendo i ritorni a capo \r\n in \n) e genera immediatamente il file `.extracted.txt` *(Figura 4.7)*.

```php
// Se è file di testo (qualsiasi text/*) creiamo il .extracted.txt con lo stesso contenuto
$basename_noext = pathinfo($savedName, PATHINFO_FILENAME);
$extractedPath = dirname($target_path) . '/' . $basename_noext . '.extracted.txt';

if (strpos($detected_mime, 'text/') === 0) {
    $txt = @file_get_contents($target_path);
    if ($txt === false) {
        error_log("Failed to read uploaded text file {$target_path}\n", 3, $EXEC_LOG);
    } else {
        // Normalizza CRLF -> LF
        $txt_clean = str_replace(["\r\n", "\r"], "\n", $txt);
        if (@file_put_contents($extractedPath, $txt_clean) === false) {
            error_log("Failed to write extracted file {$extractedPath}\n", 3, $EXEC_LOG);
        } else {
            error_log("Created extracted file for text upload: {$extractedPath}\n", 3, $EXEC_LOG);
        }
    }
}
```

Figura 4.7: Caso file di testo

4.2.4 Registrazione nel Database

Prima di avviare le operazioni di estrazione, il sistema deve inserire il record del documento associato nel database *(Figura 4.8)*.

```php
try {
    $pdo = new PDO($dbDsn, $dbUser, $dbPass, [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
        PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8mb4"
    ]);
    $check = $pdo->prepare("SELECT ID FROM Info_Tecniche WHERE Nome_File = :nome LIMIT 1");
    $check->execute([':nome' => $nomeFileArg]);
    $existing = $check->fetch(PDO::FETCH_ASSOC);
    if ($existing && !empty($existing['ID'])) {
        $docId = intval($existing['ID']);
        error_log("Found existing record for Nome_File={$nomeFileArg}, reusing ID={$docId}\n", 3, $EXEC_LOG);
    } else {
        // Non esiste: procedi con l'inserimento via cli.py
        error_log("About to call cli.py insert with nome_file={$nomeFileArg}\n", 3, $EXEC_LOG);

        // Costruisce il comando per il terminale
        $cliDir = dirname($cliPath);
        $cmdInsert = 'cd ' . escapeshellarg($cliDir)
            . ' && ' . escapeshellcmd($pythonBin)
            . ' ' . escapeshellarg($cliPath)
            . ' insert'
            . ' --nome_file ' . escapeshellarg($nomeFileArg)
            . ' --tipo ' . escapeshellarg($tipoPerInsert)
            . ' --riassunto ' . escapeshellarg('Uploaded via PHP')
            . ' 2>&1';

        exec($cmdInsert, $outInsert, $retInsert);
        $outStr = implode("\n", $outInsert);
        error_log("CMD_INSERT: {$cmdInsert}\nRET: {$retInsert}\nOUT: {$outStr}\n", 3, $EXEC_LOG);
```

Figura 4.8: Controllo di idempotenza e invocazione dello script CLI.

Tramite una query preventiva (SELECT ID ... LIMIT 1) il sistema verifica se il file sia già noto: qualora Telegram dovesse ritentare l'invio del *webhook*, lo script riutilizzerebbe l'ID, evitando la creazione di record duplicati.

Qualora il file sia nuovo, lo script richiama `cli.py` per l'inserimento iniziale del documento. Nella costruzione del comando da terminale (`$cmdInsert`), viene utilizzato `escapeshellarg()` per prevenire attacchi di *Command Injection*.

4.2.5 Elaborazione asincrona

Questa fase costituisce l'elemento fondamentale dell'architettura del processo di acquisizione dei documenti.

Le API di Telegram impongono vincoli rigidi: se il server che riceve il *webhook* non risponde con successo entro pochi secondi, la richiesta viene considerata fallita e Telegram tenta nuovamente l'invio in loop. L'estrazione ottica e

l'invocazione dei modelli linguistici non possono avvenire in maniera sincrona; di conseguenza, la soluzione adottata consiste nel delegare il carico computazionale a un processo figlio (`process_summary.php`) *(Figura 4.9)*.

```php
// Lancia process_summary in BACKGROUND
// PASSIAMO --chat_id e mettiamo TELEGRAM_BOT_TOKEN solo per questo processo figlio
$phpCliSafe = escapeshellcmd($phpCli);
$procScriptArg = escapeshellarg($processScript);
$docIdArg = escapeshellarg($docId);
$fullpathArg = escapeshellarg($target_path);
$chatIdArg = escapeshellarg($chat_id);

// imposta variabile d'ambiente solo per questo comando
$botTokenEnv = 'TELEGRAM_BOT_TOKEN=' . escapeshellarg($BOT_TOKEN);

$bgCmd = sprintf(
    '%s %s %s --id %s --fullpath %s --chat_id %s > /dev/null 2>&1 & echo $!',
    $botTokenEnv,
    $phpCliSafe,
    $procScriptArg,
    $docIdArg,
    $fullpathArg,
    $chatIdArg
);

error_log("Spawn bgCmd: {$bgCmd}\n", 3, $EXEC_LOG);
$pid = trim(shell_exec($bgCmd));
error_log("Launched process_summary in background (pid={$pid}) for ID={$docId}\n", 3, $EXEC_LOG);

// Rispondi immediatamente all'utente (evitiamo retry Telegram)
send_telegram($BOT_TOKEN, 'sendMessage', [
    'chat_id' => $chat_id,
    'text' => "Documento ricevuto e messo in coda per l'elaborazione.\nID: {$docId}\nFile: " . basename($target_path)
]);

http_response_code(200);
echo json_encode(['ok'=>true,'id'=>$docId,'queued' => true, 'pid' => $pid]);
exit;
```

Figura 4.9: Disaccoppiamento asincrono

Il comando inviato al sistema operativo è costruito tramite `sprintf()`. La funzione reindirizza lo *standard output* e lo *standard error* del processo figlio verso un dispositivo nullo (`/dev/null`), scollegandolo fisicamente dal flusso dell'interfaccia web. L'operatore `&` forza l'esecuzione in background, mentre l'istruzione `echo $!` restituisce a PHP il *Process ID*. Dopo aver lanciato il processo (`shell_exec($bgCmd)`), viene inviata all'utente una notifica su Telegram che informa che il documento è stato ricevuto ed è in fase di elaborazione. Infine, viene chiusa la connessione con Telegram, lasciando il sistema pronto ad accettare nuovi caricamenti.

4.3 Motore di estrazione

Il file `worker_extract.py` gestisce il processo di estrazione testuale dai documenti ricevuti. Il codice analizza il file e ne estrae il contenuto sfruttando librerie di sistema in base al formato del documento.

4.3.1 Inizializzazione

Per garantire la portabilità del codice, viene evitata la pratica dell'hardcoding dei percorsi. Sfruttando il modulo `os`, lo script interroga le variabili d'ambiente (caricate nel file `.env`) per individuare la posizione degli eseguibili (pdftotext, pdfimages, pdftoppm e tesseract). In questa fase viene anche definita la costante `MIN_TEXT_LENGTH`, una soglia importante che stabilisce la quantità minima di caratteri per considerare un PDF "testuale", al di sotto il sistema utilizza l' OCR.

L'interfacciamento avviene tramite una funzione di *wrapper* centralizzata `run_cmd()` che utilizza la libreria `subprocess` *(Figura 4.10)*.

```
def run_cmd(cmd_list):
    try:
        res = subprocess.run(cmd_list, stdout=subprocess.PIPE, stderr=subprocess.PIPE, check=False)
        return res.returncode, res.stdout, res.stderr
    except Exception as e:
        return 1, b'', str(e).encode()
```

Figura 4.10: Esecuzione dei comandi shell

La funzione accetta come input una lista di argomenti (`cmd_list`), garantendo che i parametri vengano passati in modo sicuro. Un eventuale fallimento (ad esempio, Tesseract non riesce a elaborare un'immagine) viene intercettato senza provocare il *crash*, permettendo al sistema di tentare estrazioni alternative.

4.3.2 Estrazione nativa da documenti PDF

Nella gestione dei file in formato PDF, come prima opzione si utilizza l'estrazione vettoriale tramite la libreria pdftotext. *(Figura 4.11)*.

```python
def extract_text_pdftotext(pdf_path: Path):
    """Estrarre testo da PDF con pdftotext (layout). Restituisce stringa ('' se vuoto)."""
    extracted_path = pdf_path.with_name(pdf_path.stem + '.extracted.txt')
    cmd = [PDFTOTEXT, '-layout', str(pdf_path), str(extracted_path)]
    rc, out, err = run_cmd(cmd)
    if rc != 0:
        logging.debug('pdftotext fallito rc=%s stderr=%s', rc, err.decode(errors='ignore'))
        return ''
    if not extracted_path.exists() or extracted_path.stat().st_size == 0:
        logging.debug('pdftotext generato file vuoto: %s', extracted_path)
        return ''
    return extracted_path.read_text(encoding='utf-8', errors='ignore')
```

Figura 4.11: Estrazione testo tramite la libreria pdftotext

La funzione `extract_text_pdftotext()` prepara il percorso di destinazione dove sarà caricato tutto il contenuto. In questa fase scelta più rilevante è l'inclusione del *flag* posizionale `-layout`. Le Infotecniche contengono tipicamente tabelle e questo parametro forza il motore a inserire spaziature fittizie, garantendo che il testo mantenga un senso logico per la fase successiva di generazione del riassunto.

Inoltre, il sistema non si limita a verificare il codice di uscita del processo. I PDF contenenti solo immagini vengono elaborati da pdftotext senza generare errori, producendo però un contenuto vuoto. Lo script, quindi, ispeziona la dimensione del file e, se risulta zero byte, il documento viene passato alla fase di OCR.

4.3.3 Riconoscimento Ottico (OCR)

Qualora l'estrazione nativa fallisca, o se il file in ingresso è un formato grafico (JPEG/PNG), il sistema ricorre alla *Computer Vision* utilizzando l'OCR.

Nel caso un PDF non è vettoriale, è necessario un passaggio intermedio di rasterizzazione (*Figura 4.12*).

```python
def extract_text_ocr_from_pdf(pdf_path: Path):
    """Converte pagine PDF in PNG con pdftoppm e lancia tesseract su ogni pagina."""
    if not Path(PDFTOPPM).exists() or not Path(TESSERACT).exists():
        logging.info('pdftoppm o tesseract non trovati: salto OCR per pdf.')
        return ''
    with tempfile.TemporaryDirectory() as td:
        prefix = os.path.join(td, 'page')
        cmd = [PDFTOPPM, '-png', str(pdf_path), prefix]
        rc, out, err = run_cmd(cmd)
        if rc != 0:
            logging.info('pdftoppm fallito, rc=%s stderr=%s', rc, err.decode(errors='ignore'))
            return ''
        # trova png generate
        pngs = sorted(glob.glob(os.path.join(td, 'page-*.png')) + glob.glob(os.path.join(td, 'page*.png')))
        if not pngs:
            logging.info('pdftoppm non ha generato immagini per %s', pdf_path)
            return ''
        return extract_text_ocr_from_images_list([Path(p) for p in pngs])
```

Figura 4.12: Rasterizzazione delle pagine del PDF tramite pdftoppm

La funzione `extract_text_ocr_from_pdf()` invoca lo strumento pdftoppm per convertire ogni singola pagina in un'immagine ad alta risoluzione salvata temporaneamente sul server.

Tramite `extract_text_ocr_from_images_list()` *(Figura 4.13)*, il sistema utilizza Tesseract su ogni immagine. Successivamente i frammenti di testo vengono normalizzati, decodificati in UTF-8 e concatenati in un'unica variabile.

```python
def extract_text_ocr_from_images_list(img_paths):
    """Usa tesseract su lista di immagini; concatena i risultati."""
    parts = []
    for i, img in enumerate(img_paths):
        base = str(img) + '.ocr'
        cmd = [TESSERACT, str(img), base]
        rc, out, err = run_cmd(cmd)
        txtfile = base + '.txt'
        if Path(txtfile).exists():
            parts.append(Path(txtfile).read_text(encoding='utf-8', errors='ignore'))
            try:
                os.remove(txtfile)
            except:
                pass
    return '\n'.join([p.strip() for p in parts]).strip()
```

Figura 4.13: Ciclo iterativo di estrazione OCR tramite Tesseract.

4.3.4 Logica di Routing e Orchestrazione Dinamica

Il funzionamento decisionale dell'intero script è imprentato dalla funzione `process_batch()` *(Figura 4.14)*.

```python
def process_batch(file_path: Path):
    try:
        logging.info("Inizio process_batch per %s", file_path)
        suffix = file_path.suffix.lower()
        text = ''
        image_exts = ['.jpg', '.jpeg', '.png', '.tiff', '.tif', '.bmp', '.webp']

        # CASO 1: L'utente ha caricato una foto
        if suffix in image_exts:
            logging.info("Rilevato file immagine, uso OCR diretto: %s", file_path)
            text = extract_text_ocr_from_image_file(file_path)
        # CASO 2: L'utente ha caricato un PDF (o altro)
        else:
            # primo tentativo pdftotext
            text = extract_text_pdftotext(file_path)
            if not text or len(text) < MIN_TEXT_LENGTH:
                logging.info("pdftotext ha restituito poco testo (%d). Provo OCR su PDF.", len(text) if text else 0)
                ocr_text = extract_text_ocr_from_pdf(file_path)
                if ocr_text and len(ocr_text) > len(text or ""):
                    text = ocr_text

        # tenta estrazioni immagini comunque (utile per PDF con immagini interne)
        try:
            images_dir = file_path.parent / file_path.stem / 'images'
            extract_images(file_path, images_dir)
        except Exception as e:
            logging.warning("Errore estrazione immagini: %s", e)

        cleaned = clean_text(text)
        extracted_path = file_path.with_name(file_path.stem + '.extracted.txt')

        try:
            # Crea il file .extracted.txt utilizzato per il riassunto
            extracted_path.write_text(cleaned, encoding='utf-8')
            logging.info("Salvato testo estratto in %s (len=%d)", extracted_path, len(cleaned))
        except Exception as e:
```

Figura 4.14: Algoritmo di routing dinamico

Se l'estensione corrisponde a un formato immagine, il sistema instrada direttamente l'elaborazione verso l'OCR. In caso di documenti PDF, il codice tenta prima l'estrazione vettoriale altrimenti, come descritto nel capitolo 4.3.2, ricorre all'OCR. Successivamente il codice invoca un processo dedicato all'estrazione degli eventuali schemi grafici (`extract_images()`) che vengono salvati sul *file system*.

Al termine del processo, lo script normalizza il testo tramite la funzione `clean_text()` (rimuovendo spazi superflui e uniformando il contenuto) e procede al salvataggio del contenuto nel file `[nome_file].extracted.txt`.

4.4 Logica Applicativa e Integrazione IA

Il file `process_summary.php` gestisce la fase di elaborazione testuale del documento, collegando il testo estratto con il modello LLM per la sintesi automatica.

Questo script, come prima operazione, attende che il motore di estrazione (`worker_extract.py`) produca il file di testo del contenuto e acquisisce il token di Hugging Face per l'autenticazione.

4.4.1 Frammentazione (Text Chunking)

Il modello LLM per la sintesi adottato per questo progetto utilizzato `facebook/bart-large-cnn` [25]. Il vincolo principale di questa categoria di modelli è la dimensione limitata della *Context Window*. Qualora la lunghezza del documento originale superi la soglia massima di token elaborabili, le API rifiutano la richiesta di elaborazione.

Per superare questa limitazione, è stato implementato un algoritmo di partizionamento nella funzione `chunk_text_into_paragraphs()` che scompone il testo mantenendo il senso logico del documento *(Figura 4.15)*.

```php
function chunk_text_into_paragraphs(string $text, int $maxChars = 1500): array {
    $text = trim(str_replace(["\r\n","\r"], "\n", $text));
    if ($text === '') return [];
    $paras = preg_split("/\n\s*\n/u", $text, -1, PREG_SPLIT_NO_EMPTY);
    $chunks = []; $current = '';
    foreach ($paras as $p) {
        $p = trim($p); if ($p === '') continue;
        if (mb_strlen($current . "\n\n" . $p,'UTF-8') <= $maxChars) {
            $current = $current === '' ? $p : ($current . "\n\n" . $p);
            continue;
        }
        if (mb_strlen($p,'UTF-8') > $maxChars) {
            $sentences = preg_split('/(?<=[\.\?\!])\s+/u', $p, -1, PREG_SPLIT_NO_EMPTY);
            $buf = '';
            foreach ($sentences as $s) {
                $s = trim($s); if ($s === '') continue;
                if (mb_strlen($buf . ' ' . $s,'UTF-8') <= $maxChars) {
                    $buf = $buf === '' ? $s : ($buf . ' ' . $s);
                } else {
                    if ($buf !== '') $chunks[] = $buf;
                    if (mb_strlen($s,'UTF-8') > $maxChars) {
                        $start = 0;
                        while ($start < mb_strlen($s,'UTF-8')) {
                            $chunks[] = mb_substr($s, $start, $maxChars, 'UTF-8');
                            $start += $maxChars;
                        }
                        $buf = '';
                    } else {
                        $buf = $s;
                    }
                }
            }
            if ($buf !== '') {
                if (mb_strlen($current . "\n\n" . $buf,'UTF-8') <= $maxChars) {
                    $current = $current === '' ? $buf : ($current . "\n\n" . $buf);
                } else {
                    if ($current !== '') { $chunks[] = $current; $current = $buf;} else { $chunks[] = $buf; $current = ''; }
                }
            }
        } else {
            if ($current !== '') { $chunks[] = $current; }
            $current = $p;
        }
    }
    if ($current !== '') $chunks[] = $current;
    return array_values(array_filter($chunks, function($c){ return trim($c) !== ''; }));
}
```

Figura 4.15: Algoritmo di text chunking

L'algoritmo adotta un approccio di raffinamento *Top-Down*. In prima istanza, isola i paragrafi del contenuto e li concatena in un blocco (*chunk*) temporaneo fino al raggiungimento di una soglia massima prestabilita (3000 caratteri). Nel caso in cui un singolo blocco risulti troppo esteso, la funzione procede ad un ulteriore raffinamento, frammentando il blocco in singole frasi basandosi sulla punteggiatura forte. In questo modo, si evita il troncamento di un concetto nel mezzo di una frase.

Per prevenire la perdita di coesione tra i blocchi, viene implementata una logica di *overlap* (`add_overlap()`) *(Figura 4.16)*.

```php
function add_overlap(array $chunks, $overlapChars = 250) {
    if (count($chunks) < 2) return $chunks;
    $out = [];
    for ($i=0;$i<count($chunks);$i++) {
        $chunk = $chunks[$i];
        if ($i>0) {
            $prev = $out[count($out)-1];
            $ov = mb_substr($prev, max(0, mb_strlen($prev,'UTF-8') - $overlapChars), $overlapChars, 'UTF-8');
            $chunk = $ov . "\n\n" . $chunk;
        }
        $out[] = $chunk;
    }
    return $out;
}
```

Figura 4.16: Logica di overlap

Viene estratta la parte finale di un blocco e concatenata all'inizio del successivo, permettendo di fornire al modello linguistico una contesto pregresso.

4.4.2 Interfacciamento con Hugging Face API

Ottenuti una serie di blocchi testuali coerenti, lo script invoca le API di Hugging Face per la generazione dei riassunti.

L'interazione è gestita dalla funzione **`hf_summarize`** *(Figura 4.17)*.

```php
function hf_summarize(string $text, string $model = 'facebook/bart-large-cnn', int $max_length = 150, int $timeout = 30, int $retries = 1): ?string {
    $hf_token = getenv('HF_TOKEN');
    if (!$hf_token) {
        error_log("HF_TOKEN missing");
        return null;
    }
    $text = trim($text);
    if ($text === '') return null;
    $url = "https://router.huggingface.co/hf-inference/models/".urlencode($model);
    $payload = json_encode([
        "inputs" => $text,
        "parameters" => ["max_length" => $max_length, "min_length" => max(10, (int)floor($max_length/4)), "do_sample" => false]
    ]);
    $attempt = 0;
```

Figura 4.17: Configurazione della connessione con le API di Hugging Face (parte 1).

Durante la costruzione del *payload* JSON, lo script fornisce dei parametri di controllo. Vengono definiti la lunghezza desiderata del riassunto e il parametro `do_sample = false`. Questa direttiva obbliga al modello di Intelligenza Artificiale a produrre un riassunto deterministico e conforme al testo originale.

Poiché l'utilizzo di API esterne espone il sistema a potenziali latenze o limitazioni di traffico, viene implementata una logica di tolleranza ai guasti *(Figura 4.18)*.

```
    while ($attempt <= $retries) {
        $attempt++;
        $ch = curl_init($url);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($ch, CURLOPT_POST, true);
        curl_setopt($ch, CURLOPT_POSTFIELDS, $payload);
        curl_setopt($ch, CURLOPT_HTTPHEADER, ["Authorization: Bearer {$hf_token}", "Content-Type: application/json"]);
        curl_setopt($ch, CURLOPT_TIMEOUT, $timeout);
        $resp = curl_exec($ch);
        $httpcode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
        $err = curl_error($ch);
        curl_close($ch);
        if ($resp === false) {
            if ($attempt <= $retries) { usleep(250000); continue; }
            return null;
        }
        if ($httpcode >= 400) {
            if (in_array($httpcode, [429,503]) && $attempt <= $retries) { usleep(500000); continue; }
            return null;
        }
        // Estrae il riassunto dal formato JSON risposto dall'IA
        $data = json_decode($resp, true);
        if (json_last_error() === JSON_ERROR_NONE) {
            if (is_array($data) && isset($data[0]['summary_text'])) return trim($data[0]['summary_text']);
            if (isset($data['summary_text'])) return trim($data['summary_text']);
        }
        return null;
    }
    return null;
}
```

Figura 4.18: Configurazione della connessione con le API di Hugging Face (parte 2).

In caso di codici HTTP 429 (*Too Many Requests*) o 503 (*Service Unavailable*), l'esecuzione non si interrompe, ma utilizza un meccanismo di riprova automatica dopo una sospensione temporale.

4.4.3 Logica di sintesi iterativa a piramide

Inviare decine di blocchi separati all'Intelligenza Artificiale produrrebbe riassunti disgiunti. Per ottenere un testo coeso, il sistema implementa una logica di sintesi stratificata *(Figura 4.19)*.

```
//Quick-shot: prova a riassumere tutto in un colpo solo se il documento è corto
$final_summary = null;
$clean_for_quick = mb_substr($contenutoEstratto, 0, 8000, 'UTF-8');
$quick = hf_summarize($clean_for_quick, 'facebook/bart-large-cnn', 200, 30, 1);
if ($quick && mb_strlen($quick,'UTF-8') > 30) {
    $final_summary = $quick;
    error_log("Quick summary successful for id={$docId} elapsed=" . round(microtime(true)-$start_time,2));
} else {
    $chunks = chunk_text_into_paragraphs($contenutoEstratto, 3000);
    $chunks = add_overlap($chunks, 200);
    $final_summary = iterative_summarize($chunks, 'facebook/bart-large-cnn', 4, 1024, 200);
}

// fallback se tutto fallisce: prende 600 char
if ($final_summary === null) {
    $final_summary = mb_substr(trim(preg_replace('/\s+/', ' ', $contenutoEstratto)), 0, 600);
}
```

Figura 4.19: Tentativo di sintesi istantanea e successiva elaborazione gerarchica.

Prima di avviare il partizionamento completo, se il contenuto estratto è sufficientemente breve (8000 caratteri) viene inviato integralmente in un'unica richiesta, in questo caso il processo termina istantaneamente.

Se il contenuto è esteso, viene demandato alla funzione `iterative_summarize()`, che segue un metodo in cui i riassunti vengono progressivamente aggregati fino al risultato finale. *(Figura 4.20)*.

```php
function iterative_summarize(array $chunks, string $model='facebook/bart-large-cnn', int $chars_per_toke
    $mini_summaries = [];
    foreach ($chunks as $c) {
        $c = trim($c);
        if ($c === '') continue;
        $s = hf_summarize($c, $model, 60, 60, 2);
        if ($s && trim($s) !== '') { $mini_summaries[] = trim($s); continue; }
        $half = mb_substr($c,0,(int)floor(mb_strlen($c,'UTF-8')/2),'UTF-8');
        if (mb_strlen($half,'UTF-8') > 200) {
            $s2 = hf_summarize($half, $model, 60, 60, 2);
            if ($s2 && trim($s2) !== '') { $mini_summaries[] = trim($s2); continue; }
        }
    }
    if (count($mini_summaries) === 0) return null;
    // riduzione gerarchica
    $max_input_chars = max(500, ($max_input_tokens - $reserve_output_tokens) * $chars_per_token);
    $current = $mini_summaries;
    $iteration = 0; $max_iterations = 6;
    while (count($current) > 1 && $iteration < $max_iterations) {
        $iteration++;
        $batches = []; $buf = '';
        foreach ($current as $s) {
            if ($buf === '') $buf = $s;
            else {
                if (mb_strlen($buf . "\n\n" . $s,'UTF-8') <= $max_input_chars) $buf .= "\n\n" . $s;
                else { $batches[] = $buf; $buf = $s; }
            }
        }
        if ($buf !== '') $batches[] = $buf;
        $next = [];
        foreach ($batches as $b) {
            $mid = hf_summarize($b, $model, 120, 60);
            if ($mid && trim($mid) !== '') $next[] = trim($mid);
            else $next[] = mb_substr($b, 0, $max_input_chars, 'UTF-8');
        }
        $current = $next;
    }
    $final_text = implode("\n\n", $current);
    $max_final_input_chars = ($max_input_tokens - $reserve_output_tokens) * $chars_per_token;
    if (mb_strlen($final_text,'UTF-8') > $max_final_input_chars) {
        $final_text = mb_substr($final_text, 0, $max_final_input_chars, 'UTF-8');
    }
    $final = hf_summarize($final_text, $model, 200, 90);
    if ($final && trim($final) !== '') return trim($final);
    return mb_substr($final_text, 0, 1024, 'UTF-8');
}
```

Figura 4.20: Architettura per la riduzione dei blocchi di testo.

La funzione elabora un mini-riassunto per ciascun blocco originario. Successivamente, avvia una fase di riduzione iterativa: i riassunti parziali vengono concatenati per formare blocchi più densi, i quali vengono nuovamente inviati al modello. Questo ciclo viene ripetuto finché i blocchi iniziali non formano un unico riassunto finale.

4.4.4 Aggiornamento dati

Al termine del riassunto finale generato, si ripristina la comunicazione con il database per memorizzare i dati *(Figura 4.21)*.

```php
try {
    $dsn = "mysql:host=$dbHost;dbname=$dbName;charset=utf8mb4";
    $pdo = new PDO($dsn, $dbUser, $dbPass, [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]);
    $upd = $pdo->prepare("UPDATE Info_Tecniche SET Riassunto = :riassunto, Contenuto = :contenuto, Numero = :numero WHERE ID = :id");
    $upd->execute([
        ':riassunto' => $final_summary,
        ':contenuto' => $contenutoEstratto,
        ':numero' => $numero,
        ':id' => $docId
    ]);
    error_log("OK updated ID={$docId} elapsed=" . round(microtime(true)-$start_time,2));

    // Invia messaggio Telegram al chat_id
    if (!empty($chatId)) {
        $msg = "Documento elaborato.\nID: {$docId}\nFile: " . basename($pdfPath) . "\n\nRiassunto:\n" . mb_substr($final_summary, 0, 3
        $res = send_telegram_message($chatId, $msg);
        if ($res === false) {
            error_log("Failed to send Telegram message for ID={$docId}");
        }
    }

    echo "OK updated ID={$docId}\n";
} catch (Exception $e) {
    fwrite(STDERR, "DB update error: " . $e->getMessage() . "\n");
    if (!empty($chatId)) {
        send_telegram_message($chatId, "Errore nell'aggiornamento DB per ID {$docId}: " . $e->getMessage());
    }
    exit(4);
}

// Rimuovi il file .extracted.txt per evitare accumulo
try {
    if (file_exists($extractedFile)) {
        unlink($extractedFile);
    }
} catch (Exception $e) {
    fwrite(STDERR, "Warning: impossible to unlink extracted file: " . $e->getMessage() . "\n");
}
```

Figura 4.21: Aggiornamento dati nel DB, feedback all'utente e housekeeping del file system.

Il record viene aggiornato tramite costrutti di tipo *Prepared Statement*, che separano le istruzioni SQL dai parametri, neutralizzando ogni rischio di *SQL Injection*.

Infine, viene inviato all'utente un messaggio su Telegram di un'anteprima del riassunto confermando l'indicizzazione del documento e conclude le operazioni eliminando il file di testo temporaneo del contenuto (`.extracted.txt`).

4.5 Comunicazione Inter-Processo

L'architettura del sistema si basa su una separazione delle responsabilità tra il *routing* di rete in PHP e il motore computazionale in Python. Di conseguenza, si è reso necessario realizzare un canale di comunicazione tra i due ambienti implementato tramite il file `cli.py`.

4.5.1 Inserimento record

La funzione `insert_record()` gestisce la registrazione iniziale dei documenti nel database. *(Figura 4.22)*.

```python
def insert_record(nome_file, tipo='application/pdf', riassunto='Uploaded via PHP'):
    """Insert row into DB and return new id."""
    conn = get_conn()
    try:
        with conn.cursor() as cur:
            sql = """INSERT INTO Info_Tecniche (Tipo, Nome_File, Contenuto, Riassunto)
                     VALUES (%s, %s, %s, %s)"""
            cur.execute(sql, (tipo, nome_file, '', riassunto))
            # Se conn.autocommit è True non serve commit; altrimenti:
            try:
                conn.commit()
            except Exception:
                pass
            nid = cur.lastrowid
    finally:
        conn.close()
    return nid
```

Figura 4.22: Inserimento record nel database

Nell'inserimento, come già illustrato nei capitoli precedenti, per garantire la sicurezza vengono utilizzate query parametrizzate e il passaggio dei parametri avviene tramite *binding* posizionale neutralizza il rischio di *SQL Injection*.

Al termine dell'inserimento e del consolidamento dei dati (`commit`), il sistema recupera l'ID della chiave primaria assegnato da MySQL e lo restituisce, assicurandosi di rilasciare le risorse chiudendo la connessione al database.

4.5.2 Interfaccia a riga di comando

Il file `cli.py` necessita di interpretare quale specifica operazione sia stata richiesta dal server web (inserire un documento, avviare una ricerca, recuperare i dettagli di un record). Per gestire questa dinamica, lo script si avvale della libreria **`argparse`** *(Figura 4.23)*.

```python
def main():
    parser = argparse.ArgumentParser(description='CLI bridge for PHP -> Python')
    sub = parser.add_subparsers(dest='cmd')

    p = sub.add_parser('search')
    p.add_argument('--q', required=True)
    p.add_argument('--limit', type=int, default=30)
    p.add_argument('--offset', type=int, default=0)
    p.add_argument('--mode', choices=['natural','boolean'], default='natural')
    p.add_argument('--page', type=int)
    p.add_argument('--per_page', type=int)

    p = sub.add_parser('doc')
    p.add_argument('--id', required=True, type=int)

    p = sub.add_parser('insert')
    p.add_argument('--nome_file', required=True)
    p.add_argument('--tipo', default='application/pdf')
    p.add_argument('--riassunto', default='Uploaded via PHP')

    args = parser.parse_args()
    if not args.cmd:
        parser.print_help()
        return 1

    return globals().get(f'cmd_{args.cmd}')(args)

if __name__ == '__main__':
    sys.exit(main())
```

Figura 4.23: Configurazione dei sub-parsers e logica di routing

Vengono definiti i comandi (`search, doc, insert`), ciascuno associato alla sua specifica funzionalità. In questa architettura la validazione degli input è demandata al modulo argparse: qualora vengano omessi parametri o forniti valori non conformi, l'esecuzione viene interrotta. L'instradamento delle richieste è realizzato mediante un meccanismo di risoluzione dinamica, attraverso l'istruzione **`globals().get(f'cmd_{args.cmd}')(args)`**. A partire dal nome del sottocomando richiesto, il sistema costruisce il nome della

funzione da invocare aggiungendo il prefisso `cmd_` e cerca nei nomi del programma la funzione corrispondente da invocare.

4.5.3 Metodi comunicazione Inter-Processo

Per il passaggio di dati strutturati tra processi isolati, il sistema sfrutta lo *standard output* del terminale Linux come canale di comunicazione primario, adottando il formato JSON come standard.

I metodi di elaborazione incapsulano i risultati in dizionari strutturati *(Figura 4.24)*.

```python
def cmd_search(args):
    # normalizza valori
    limit = int(args.limit or 30)
    offset = int(getattr(args, 'offset', 0) or 0)
    rows = safe_search(args.q, limit=limit, offset=offset, boolean=(args.mode=='boolean'))
    out = []
    for r in rows:
        out.append({
            'ID': r.get('ID'),
            'Nome_File': r.get('Nome_File'),
            'Numero': r.get('Numero') or '',
            'score': float(r.get('score')) if r.get('score') is not None else 0.0,
            'Riassunto': r.get('Riassunto') or '',
        })
    print(json.dumps(out, ensure_ascii=False))
    return 0

def cmd_doc(args):
    """
    Restituisce il dettaglio di un documento come JSON.
    """
    try:
        conn = get_conn()
        try:
            with conn.cursor() as cur:
                cur.execute("SELECT ID, Nome_File, Numero, Contenuto, Riassunto FROM Info_Tecniche WHERE ID = %s", (args.id,))
                row = cur.fetchone()
        finally:
            conn.close()
    except Exception as e:
        # ritorna JSON d'errore e codice 2 per compatibilità con chiamata CLI
        print(json.dumps({'error': str(e)}))
        return 2

    if not row:
        print(json.dumps({'error': 'not found'}))
        return 3

    # assicura output unicode leggibile
    print(json.dumps(row, ensure_ascii=False))
    return 0

def cmd_insert(args):
    try:
        nid = insert_record(args.nome_file, tipo=args.tipo, riassunto=args.riassunto)
        print(json.dumps({'insert_id': nid}))
        return 0
    except Exception as e:
        print(json.dumps({'error': str(e)}))
        return 5
```

Figura 4.24: Metodi di elaborazione

Durante la conversione in JSON, viene utilizzato il parametro `ensure_ascii=False`. Questa configurazione disabilita l'escaping dei

caratteri non ASCII e preserva la codifica UTF-8. In questo modo vengono mantenute lettere accentate, segni di punteggiatura e simboli tecnici.

Ogni funzione termina restituendo un valore numerico che identifica l'esito dell'operazione. Lo script Python intercetta l'eccezione, genera un oggetto JSON contenente i dettagli dell'errore e restituisce un codice di stato specifico. In questo modo, PHP in ascolto decodifica l'errore e intraprende le necessarie azioni di *fallback*.

4.6 Motore di ricerca testuale

Il file `searcher.py` rappresenta il modulo dedicato alla ricerca. Al suo interno sono implementate le funzioni per interrogare il database e fornire i risultati agli altri componenti del sistema.

4.6.1 Interrogazione indice Full-Text

La funzione principale `search()`, si occupa di tradurre i termini inseriti dall'utente in query ottimizzate. Questo script sfrutta un indice *Full-Text* applicato al contenuto dei documenti (*Figura 4.25*).

```
def search(query, limit=30, offset=0, boolean=False):
    if not query or not query.strip():
        return []

    conn = get_conn()
    try:
        with conn.cursor() as cur:
            if boolean:
                sql = """
                SELECT ID, Nome_File, Numero, Riassunto,
                       MATCH(Contenuto) AGAINST (%s IN BOOLEAN MODE) AS score
                FROM Info_Tecniche
                WHERE MATCH(Contenuto) AGAINST (%s IN BOOLEAN MODE)
                ORDER BY score DESC
                LIMIT %s OFFSET %s
                """
            else:
                sql = """
                SELECT ID, Nome_File, Numero, Riassunto,
                       MATCH(Contenuto) AGAINST (%s) AS score
                FROM Info_Tecniche
                WHERE MATCH(Contenuto) AGAINST (%s)
                ORDER BY score DESC
                LIMIT %s OFFSET %s
                """
            cur.execute(sql, (query, query, int(limit), int(offset)))
            rows = cur.fetchall()
    finally:
        conn.close()
    return rows or []
```

Figura 4.25: Logica di interrogazione Full-Text

Questa implementazione introduce un paradigma di ricerca basato sulla rilevanza statistica. Il motore di ricerca valuta molteplici fattori: la frequenza di comparsa del termine, la lunghezza complessiva del testo e la rarità della

parola rispetto all'intero contenuto. L'algoritmo restituisce quindi un punteggio di rilevanza (`score`) per ogni record. Lo script forza l'ordinamento decrescente basato su questo valore (`ORDER BY score DESC`), garantendo all'utente la visualizzazione prioritaria dei documenti più pertinenti.

La funzione prevede inoltre due strategie di interrogazione, governate da un parametro booleano. La modalità predefinita è la ricerca in linguaggio naturale, che valuta la pertinenza della query ignorando termini poco significativi, come articoli e preposizioni. In alternativa, la ricerca booleana consente di utilizzare operatori logici (ad esempio + e -) per includere o escludere termini tecnici, così di eseguire interrogazioni più restrittive.

4.6.2 Paginazione e Wrapper di Sicurezza

La funzione `search()` implementa nativamente un meccanismo di paginazione *server-side*. Tramite parametri numerici (`limit` e `offset`) per la paginazione, il sistema interroga il database a estrarre il numero di righe della tabella necessario a popolare la schermata.

Per preservare la stabilità dell'infrastruttura, l'interazione tra l'interfaccia a riga di comando (`cli.py`) e il motore di ricerca non è diretta, ma incapsulata nella funzione `safe_search()` (*Figura 4.26*).

```
def safe_search(q, limit=30, offset=0, boolean=False):
    """Call search() with fallback if signature differs."""
    try:
        return search(q, limit=limit, offset=offset, boolean=boolean)
    except TypeError:
        rows = search(q, limit=limit, boolean=boolean)
        if offset:
            return rows[offset:offset+limit]
        return rows
```

Figura 4.26: Metodo safe_search() del file cli.py

Questa funzione applica un principio di tolleranza ai guasti: tenta prima di invocare il metodo passando l'intero set di parametri ottimali. Qualora l'interprete Python sollevi un'eccezione il *wrapper* intercetta l'errore senza provocare il blocco dell'applicativo.

4.7 Interfaccia Web

L'interfaccia web utilizzata dagli utenti per la consultazione delle Infotecniche, `index.html`, è stata progettata seguendo il paradigma delle *Single Page Application* (SPA). Questo approccio prevede il caricamento della pagina una sola volta; le interazioni sono gestite in maniera asincrona da JavaScript, senza alcun ricaricamento. L'ambiente si appoggia alla libreria jQuery per la manipolazione del DOM e le chiamate di rete, e al plugin DataTables per la gestione della griglia dati.

4.7.1 Data-Grid e ricerca asincrona

L'elemento principale dell'interfaccia è costituito da una tabella che, al caricamento, viene convertita in un *Data-Grid* interattivo. Una scelta significativa risiede nell'attivazione del parametro `serverSide: true` (*Figura 4.27*). Se questo valore fosse impostato a *false*, il *browser* tenterebbe di scaricare l'intero database delle Infotecniche al primo accesso. Attivando l'elaborazione *server-side*, invece, la tabella delega le operazioni al backend.

```
<script>
    $(document).ready(function(){
        var table = $('#tabella_info').DataTable({
            serverSide: true,
            ajax: 'api/datatables_search.php',
            processing: true,
            pageLength: 10,
            responsive: true,
            autoWidth: false,
            scrollX: true,
            language: {
                url: 'https://cdn.datatables.net/plug-ins/1.13.6/i18n/it-IT.json'
            },
            columns: [
                {data:'ID', orderable: true},
                {data:'Nome_File', orderable: false},
                {data:'Numero', orderable: true},
                {data:'Riassunto', orderable: false},
                {data:'Azioni', orderable:false, searchable:false}
            ],
            order: [[0, 'asc']]
        });
```

Figura 4.27: Inizializzazione libreria DataTables in modalità server-side

Quando l'utente interagisce con l'interfaccia (digitando una parola chiave o sfogliando le pagine), il plugin intercetta l'evento, e viene inviata la richiesta HTTP GET indirizzata all'endpoint `datatables_search.php`. Non appena restituisce il dizionario JSON, la tabella associa i dati delle rispettive colonne, aggiornando solo i record a schermo.

4.7.2 Ottimizzazione di rete e caching locale

Quando un utente clicca sul bottone "Azioni" per visionare un'Infotecnica, il sistema necessita di recuperare i dati del documento che non sono stati inclusi nel *payload* iniziale.

Per ridurre i tempi di attesa, il recupero dei dettagli è gestito mediante una logica di caching lato client *(Figura 4.28)*.

```
$('#tabella_info tbody').on('click', '.btn-open', function(){
    var id = $(this).data('id');
    if (!id) return alert('ID mancante');

    var cacheKey = 'doc_' + id;
    var cached = sessionStorage.getItem(cacheKey);
    if (cached) return renderDoc(JSON.parse(cached));

    $('#modalTitle').text('Caricamento...');
    $('#modalSummary').text('');
    $('#modalContent').text('Caricamento contenuto…');
    $('#modalPdfLink').hide();
    $('#modalImages').empty();
    $('#docModal').show();

    $.getJSON('api/view_doc.php', {id: id})
    .done(function(res){
        if (res.error) { alert('Errore: ' + res.error); return; }
        try { sessionStorage.setItem(cacheKey, JSON.stringify(res)); } catch(e){}

        renderDoc(res);
    })
    .fail(function(xhr){
        alert('Errore server: ' + xhr.status);
        $('#docModal').hide();
    });
});
```

Figura 4.28: Gestione degli eventi e caching locale

Avvalendosi delle API Web messe a disposizione del browser, il sistema verifica la presenza dei dati nella cache locale utilizzando l'identificativo del documento e inoltra una richiesta all'endpoint `view_doc.php`. Completata la

richiesta, il JSON restituito viene elaborato nella cache locale, ottimizzando le successive consultazioni per la risorsa.

4.7.3 Rendering dinamico

Una volta che i dati del documento, l'elaborazione è affidata alla funzione `renderDoc()` incaricata di tradurre l'oggetto JSON in elementi visivi e popolare una finestra modale sulla pagina principale. (*Figura 4.29*).

```
function renderDoc(res) {
    $('#modalTitle').text(res.Nome_File || 'Dettaglio');
    $('#modalSummary').text(res.Riassunto || '');
    $('#modalContent').text(res.Contenuto || res.Riassunto || 'Nessun contenuto disponibile');

    var nome = (res.Nome_File || '').toString();
    var ext = '';
    var m = nome.match(/\.([^.\/\\]+)$/);
    if (m) ext = m[1].toLowerCase();

    if (ext === 'txt') {
        $('#modalPdfLink').hide();
    } else {
        if (res.pdf_url) {
            $('#modalPdfLink').attr('href', res.pdf_url).show();
        } else {
            $('#modalPdfLink').hide();
        }
    }
```

Figura 4.29: Rendering dinamico della vista di dettaglio

Il modulo di *rendering* implementa una logica basata sull'estensione del file: se il formato è di tipo testuale, il link di download viene nascosto all'utente; negli altri casi, invece, viene visualizzato.

4.8 Endpoint di ricerca asincrona

Affinché la griglia dei dati possa operare, la libreria `DataTables` necessita di un canale di comunicazione bidirezionale con il *server*. Questo ruolo è ricoperto dal file `datatables_search.php`, un endpoint progettato per intercettare le richieste asincrone del client, gestire le ricerche del contenuto dei documenti e restituire un *payload* JSON.

4.8.1 Interfacciamento CLI

Una volta che i parametri della richiesta sono stati validati, il codice delega la ricerca testuale a `cli.py`, costruendo e lanciando un comando eseguibile nel terminale del *server* (*Figura 4.30*).

```
$python = '/usr/bin/python3';
$cli = __DIR__ . '/../cli.py';
$limit = (int)$length;
$offset = (int)$start;
$escaped_q = escapeshellarg($q);
$mode_flag = ($mode === 'boolean') ? 'boolean' : 'natural';

$cmd = escapeshellcmd($python) . ' ' . escapeshellarg($cli) . ' search --q ' . $escaped_q . ' --limit ' . escapeshellarg($limit) .
dbg("EXEC CMD: " . $cmd);

exec($cmd, $out, $ret);
$outStr = implode("\n", $out);
dbg("CLI ret={$ret} out_preview=" . substr($outStr,0,2000));

$parseCandidate = trim($outStr);
if (!($parseCandidate && (substr($parseCandidate,0,1)=='[' || substr($parseCandidate,0,1)=='{'))) {
    for ($i=count($out)-1;$i>=0;$i--){
        $line = trim($out[$i]);
        if ($line && (substr($line,0,1)=='[' || substr($line,0,1)=='{')) { $parseCandidate = $line; break; }
    }
}

$data_cli = json_decode($parseCandidate, true);
```

Figura 4.30: Comando shell e algoritmo di parsing JSON

Lo script sanifica l'intero contesto di esecuzione: isola il percorso fisico degli eseguibili e neutralizza ogni singolo parametro dinamico.

Dopo l'esecuzione del comando, l'interfaccia web attende la risposta del terminale. A questo punto, invece di assumere che la risposta sia perfettamente formattata, il codice implementa un algoritmo di estrazione retroattiva. Lo script cicla l'array dei risultati partendo dall'ultimo elemento fino al primo; non appena individua il carattere di apertura tipico di un formato JSON (parentesi quadra o graffa), identifica la stringa relativa come il *payload* dati effettivo, lo decodifica e lo restituisce al frontend, scartando i residui del terminale.

4.9 Endpoint di dettaglio del documento

L'ultima parte dell'architettura di backend è rappresentato dal file `view_doc.php`. Questo script funge da ponte tra la griglia dei dati e la finestra modale di dettaglio di un documento.

4.9.1 Esecuzione inter-Processo e normalizzazione

Dopo la messa in sicurezza dell'ambiente operativo e sulla validazione della richiesta in ingresso, lo script procede al recupero dei dati (doc --id), delegando l'operazione al file `cli.py`.

Per garantire la stabilità del servizio e renderlo immune a variazioni strutturali, il sistema implementa una logica di normalizzazione convertendo forzatamente tutte le chiavi in minuscolo.

4.9.2 Risoluzione dei percorsi

L'ultima fase dello script prepara il *payload* JSON da restituire al client, assemblando il nome del file, il riassunto e il testo integrale. L'operazione più importante riguarda la generazione dell'URL per il download del file PDF (*Figura 4.31*).

```php
$nome_file = $normalized['nome_file'] ?? ($data['Nome_File'] ?? '');
$riassunto = $normalized['riassunto'] ?? ($data['Riassunto'] ?? '');
$contenuto = $normalized['contenuto'] ?? ($data['Contenuto'] ?? '');

$public_pdf_base = '/storage/uploads/';
$pdf_url = $nome_file ? rtrim($public_pdf_base, '/') . '/' . rawurlencode($nome_file) : null;

$images = $normalized['images'] ?? ($data['images'] ?? null);

$result = [
    'ID' => intval($normalized['id'] ?? ($data['ID'] ?? $id)),
    'Nome_File' => $nome_file,
    'Riassunto' => $riassunto,
    'Contenuto' => $contenuto,
    'pdf_url' => $pdf_url,
];
```

Figura 4.31: Risoluzione dinamica dei percorsi di rete

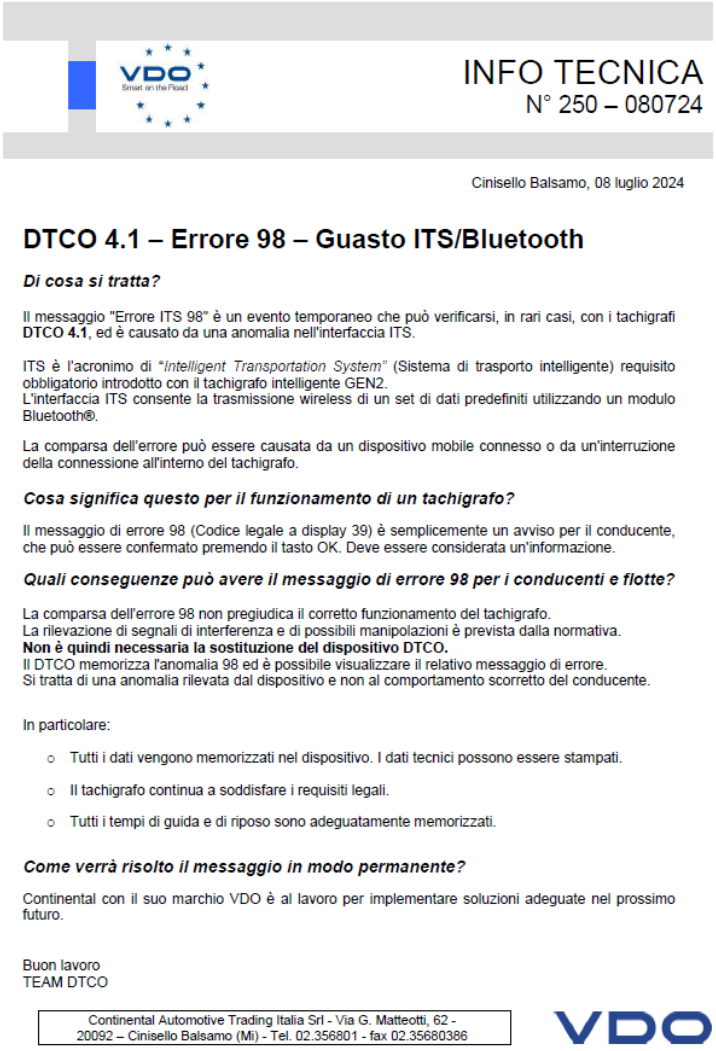
I file caricati risultano con nomi complessi, qualora venissero concatenati direttamente al percorso *base* del server, l'URL risultante violerebbe gli standard di indirizzamento http. Il problema è risolto tramite la funzione `rawurlencode()`, che processa la stringa e converte i caratteri non

alfanumerici nella rispettiva codifica, rendendo l'URL formalmente valido per i browser.

4.10 Risultato Finale

La pagina web è stata rilasciata in ambiente di produzione ed è accessibile all'indirizzo https://infotecniche.myiso.it/index.html. Tuttavia, l'interfaccia attualmente online ha subito variazioni grafiche successive alla conclusione del periodo di tirocinio. Le immagini seguenti documentano la versione consegnata al termine del progetto, illustrandone il funzionamento attraverso un caso d'uso reale.

Il punto di partenza del flusso è il documento originale. A titolo di esempio, si prenda Infotecnica in formato PDF (*Figura 4.41*).



INFO TECNICA
N° 250 – 080724

Cinisello Balsamo, 08 luglio 2024

DTCO 4.1 – Errore 98 – Guasto ITS/Bluetooth

Di cosa si tratta?

Il messaggio "Errore ITS 98" è un evento temporaneo che può verificarsi, in rari casi, con i tachigrafi **DTCO 4.1**, ed è causato da una anomalia nell'interfaccia ITS.

ITS è l'acronimo di "*Intelligent Transportation System*" (Sistema di trasporto intelligente) requisito obbligatorio introdotto con il tachigrafo intelligente GEN2.
L'interfaccia ITS consente la trasmissione wireless di un set di dati predefiniti utilizzando un modulo Bluetooth®.

La comparsa dell'errore può essere causata da un dispositivo mobile connesso o da un'interruzione della connessione all'interno del tachigrafo.

Cosa significa questo per il funzionamento di un tachigrafo?

Il messaggio di errore 98 (Codice legale a display 39) è semplicemente un avviso per il conducente, che può essere confermato premendo il tasto OK. Deve essere considerata un'informazione.

Quali conseguenze può avere il messaggio di errore 98 per i conducenti e flotte?

La comparsa dell'errore 98 non pregiudica il corretto funzionamento del tachigrafo.
La rilevazione di segnali di interferenza e di possibili manipolazioni è prevista dalla normativa.
Non è quindi necessaria la sostituzione del dispositivo DTCO.
Il DTCO memorizza l'anomalia 98 ed è possibile visualizzare il relativo messaggio di errore.
Si tratta di una anomalia rilevata dal dispositivo e non al comportamento scorretto del conducente.

In particolare:

- Tutti i dati vengono memorizzati nel dispositivo. I dati tecnici possono essere stampati.
- Il tachigrafo continua a soddisfare i requisiti legali.
- Tutti i tempi di guida e di riposo sono adeguatamente memorizzati.

Come verrà risolto il messaggio in modo permanente?

Continental con il suo marchio VDO è al lavoro per implementare soluzioni adeguate nel prossimo futuro.

Buon lavoro
TEAM DTCO

Continental Automotive Trading Italia Srl - Via G. Matteotti, 62 -
20092 – Cinisello Balsamo (Mi) - Tel. 02.356801 - fax 02.35680386

Figura 4.41: Esempio Infotecnica

Una volta caricato ed elaborato dal sistema, il documento diviene immediatamente fruibile sulla schermata principale (*Figura 4.42*).

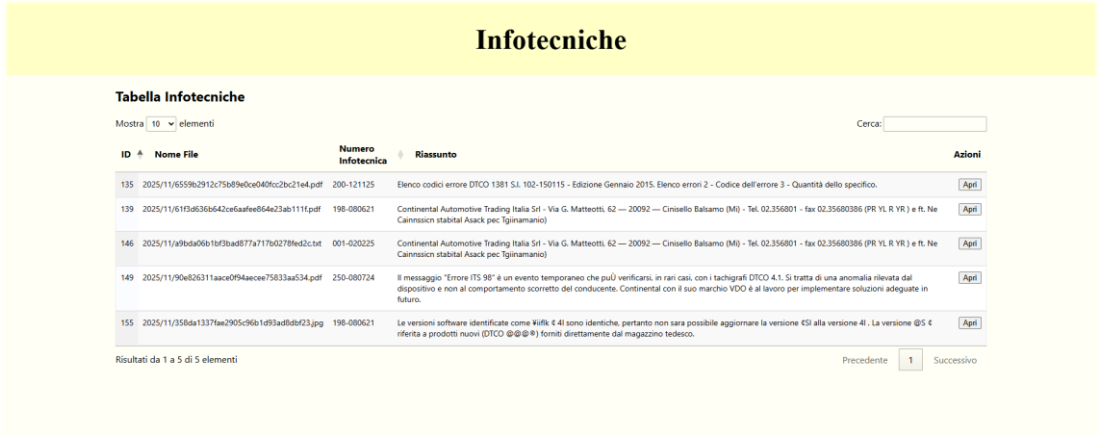


Figura 4.42: Schermata principale

Qualora l'utente necessitasse di consultare il documento, dal bottone "Apri" invoca istantaneamente la vista di dettaglio (*Figura 4.43*). La finestra modale mostra la sintesi, il testo integrale estratto e il collegamento ipertestuale per scaricare il file PDF originale.

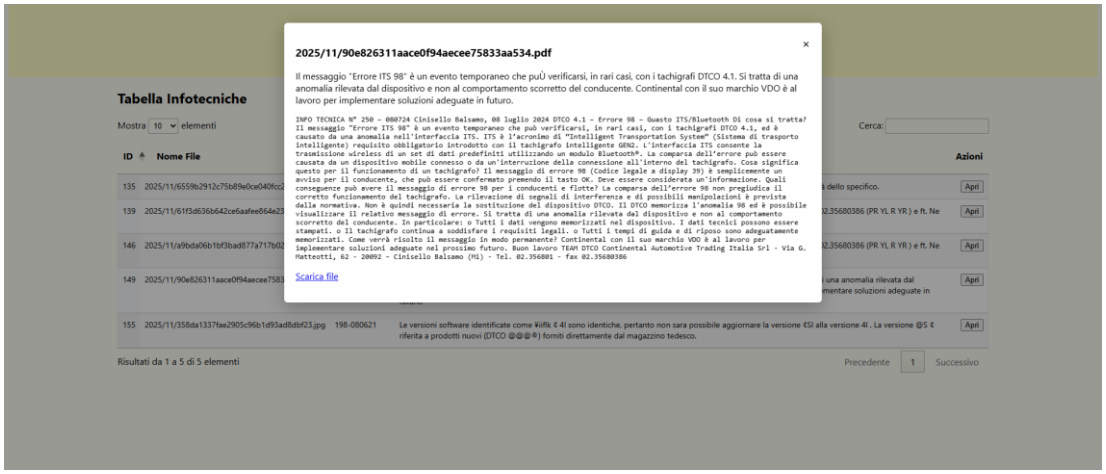


Figura 4.43: Finestra di dettaglio

Conclusioni

Il presente elaborato ha documentato il processo di sviluppo di un'architettura software dedicata all'acquisizione, elaborazione e consultazione di Infotecniche. L'obiettivo principale è stato quello di integrare l'Intelligenza Artificiale all'interno di un portale web, per automatizzare la sintesi dei documenti tecnici e fornire agli operatori uno strumento in grado di ridurre i tempi di ricerca per la risoluzione degli errori mostrati dei tachigrafi.

Il punto di partenza è stato l'analisi del contesto: la necessità tecnica di estrarre le informazioni dalle Infotecniche fornendo una sintesi fedele del contenuto. L'analisi ha portato alla progettazione e alla definizione dell'architettura di sistema. Infine, è seguita la fase di implementazione, che ha visto l'integrazione delle operazioni di estrazione, della sintesi neurale e lo sviluppo dell'intera infrastruttura web per la consultazione dei documenti.

Dal punto di vista personale, questo progetto durante il periodo di tirocinio ha rappresentato un'importante opportunità di crescita, poiché ha consentito di affrontare lo sviluppo di architetture web *full-stack*, l'amministrazione del database e l'implementazione di modelli LLM superando le sfide ingegneristiche reali nel passaggio dall'ambiente di sviluppo alla messa in produzione.

L'impiego delle tecnologie utilizzate in questo elaborato dimostra la rapida evoluzione dell'ingegneria del software orientata all'automazione. Sebbene il sistema odierno abbia innovato l'estrazione dei dati dai documenti, gli sviluppi futuri potrebbero orientarsi verso un'integrazione architetturale ancora più profonda. In particolare, l'infrastruttura intelligente potrà incrociare le istruzioni dei manuali direttamente con i codici di errore trasmessi in tempo reale dai tachigrafi. In questa prospettiva, il sistema realizzato si pone come una base di partenza per portare l'assistenza tecnica verso nuovi standard di efficienza.

Bibliografia

[1] Git, *"Git - fast, scalable, distributed revision control system"*. https://git-scm.com/

[2] GitHub Inc., *"GitHub: Let's build from here"*. https://github.com/

[3] Microsoft Corporation, *"Visual Studio Code - Code Editing. Redefined"*. https://code.visualstudio.com/

[4] Simon Tatham, *"PuTTY: a free SSH and Telnet client"*. https://www.chiark.greenend.org.uk/~sgtatham/putty/

[5] Tim Kosse et al., *"FileZilla - The free FTP solution"*. https://filezilla-project.org/

[6] Nginx Inc., *"NGINX - High Performance Load Balancer, Web Server, & Reverse Proxy"*. https://nginx.org/

[7] The PHP Group, *"PHP: Hypertext Preprocessor"*. https://www.php.net/

[8] Python Software Foundation, *"Python Language Reference"*. https://www.python.org/

[9] freedesktop.org, *"Poppler - PDF rendering library"*. https://poppler.freedesktop.org/

[10] Ray Smith, *"An Overview of the Tesseract OCR Engine"*. Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR), IEEE, 2007.

[11] Telegram FZ-LLC, *"Telegram Bot API"*. https://core.telegram.org/bots/api

[12] Hugging Face Inc., *"Hugging Face Inference API Documentation"*. https://huggingface.co/docs/api-inference/

[13] Oracle Corporation, *"MySQL 8.0 Reference Manual"*. https://dev.mysql.com/doc/

[14] Jakub Vrána, *"Adminer - Database management in a single PHP file"*. https://www.adminer.org/

[15] W3C, *"HTML5: A vocabulary and associated APIs for HTML and XHTML"*. https://html.spec.whatwg.org/

[16] W3C, *"Cascading Style Sheets (CSS)"*. Disponibile online: https://www.w3.org/Style/CSS/

[17] ECMA International, *"ECMAScript Language Specification"*. https://tc39.es/ecmascript/

[18] The jQuery Foundation, *"jQuery: The Write Less, Do More, JavaScript Library"*. https://jquery.com/

[19] ECMA International, *"The JSON Data Interchange Syntax (ECMA-404)"*. https://www.json.org/

[20] SpryMedia Ltd, *"DataTables - Table plug-in for jQuery"*. https://datatables.net/

[21] AlmaLinux OS Foundation, *"AlmaLinux OS - Forever-Free Enterprise-Grade Operating System"*. https://almalinux.org/

[22] OpenVPN Inc., *"OpenVPN - Open Source VPN server and client"*. https://openvpn.net/community/

[23] Trygve Reenskaug, *"Models-Views-Controllers"*. Xerox PARC, 1979.

[24] Daniel Stenberg et al., *"cURL - command line tool and library for transferring data with URLs"*. https://curl.se/

[25] Mike Lewis et al., "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". https://arxiv.org/abs/1910.13461