Flint

iClaudes - Savo, Favitta, Sinibaldi, Ranfoni

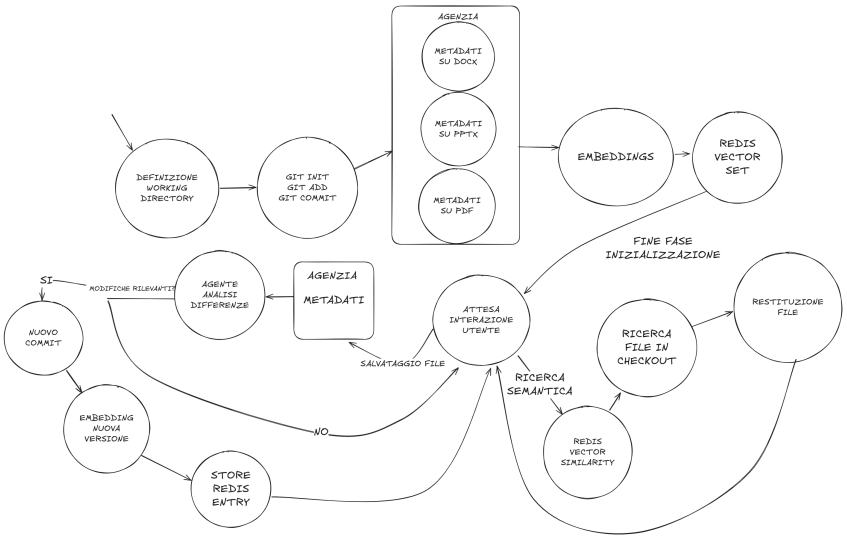
May 11, 2025

1 Introduzione

La gestione e il reperimento di file in sistemi con grandi quantità di dati possono risultare complessi. Gli utenti spesso ricordano frammenti del contenuto di un file ma faticano a localizzarlo usando metodi di ricerca tradizionali basati su nome o semplici parole chiave. Questo progetto affronta tale sfida introducendo un agente AI che (1) permette la ricerca semantica dei file e (2) traccia automaticamente le versioni dei file, consentendo una ricerca "temporale" basata sul contenuto. L'obiettivo è fornire un'interfaccia uomo-macchina più intuitiva ed efficiente per la navigazione e il recupero di informazioni, evitando ridondanti "click".

2 Architettura del Sistema

Il sistema è composto da un'applicazione frontend Electron e un backend Python, che comunicano tramite WebSockets. Redis è utilizzato per la memorizzazione degli embedding e dei metadati dei file, mentre Git gestisce il versionamento a basso livello.



2.1 Frontend (Electron)

Il frontend è costruito utilizzando Electron, fornendo un'interfaccia utente cross-platform simile a una finestra di "spotlight search".

- **main.js**: Processo principale di Electron. Gestisce il ciclo di vita dell'applicazione, la creazione della finestra (BrowserWindow), la registrazione di scorciatoie globali (es. Option+Space per mostrare/nascondere la finestra) e la comunicazione IPC con il processo di rendering. Inizializza la connessione WebSocket al backend.
- **renderer.js**: Logica del processo di rendering (UI). Gestisce l'input dell'utente, la visualizzazione dei risultati della ricerca e l'interazione con essi (es. apertura file). Comunica con il processo principale tramite l'API esposta da `preload.js`.
- **preload.js**: Script che espone in modo sicuro funzionalità selezionate di Electron (come `ipcRenderer`) al processo di rendering, mantenendo l'isolamento del contesto.

2.2 Backend (Python)

Il backend gestisce la logica di business principale, inclusa l'elaborazione dei file, la generazione di embedding, la ricerca e il versionamento.

- **app.py**: Componente centrale del backend.
  - Inizializza la connessione a Redis e carica il modello di embedding (SentenceTransformer).
  - Utilizza `watchdog` per monitorare le modifiche ai file system (creazione, modifica, cancellazione, spostamento) in una directory specificata.
  - Gestisce un repository Git nella directory monitorata, creando automaticamente commit per ogni modifica rilevante ai file (principalmente file .docx).
  - Per ogni file modificato e committato, richiama `agent.py` per estrarre il testo e generare un riassunto.
  - Genera embedding vettoriali dal riassunto del file utilizzando il modello SentenceTransformer.
  - Memorizza in Redis i metadati del file (nome, percorso relativo, hash del commit, riassunto) e il suo embedding vettoriale. L'ID del record in Redis combina il percorso relativo del file e l'hash del commit, permettendo di indicizzare diverse versioni dello stesso file.
  - Avvia un server WebSocket (implementato in `ChangeHandler`) per ascoltare le richieste di ricerca dal frontend.
- **agent.py**: Responsabile dell'estrazione del contenuto testuale da diversi tipi di file (DOCX, PDF, XLSX, PPTX) e della generazione di riassunti significativi utilizzando un modello linguistico generativo (Google Gemini). I riassunti sono progettati per catturare l'essenza del documento e includono parole chiave.
- **searcher.py**: Riceve la query dell'utente dal frontend (tramite `app.py`).
  - Utilizza Google Gemini per analizzare e "espandere" la query dell'utente, trasformandola in una descrizione ipotetica del contenuto del documento ricercato. Questo testo arricchito viene poi utilizzato per la ricerca semantica.
  - Genera un embedding per la query analizzata.
  - Esegue una ricerca di similarità vettoriale in Redis (contro gli embedding dei file indicizzati) per trovare i file/versioni più pertinenti.
  - Restituisce i percorsi dei file corrispondenti al frontend.
- **diff.py**: Fornisce funzionalità per confrontare due versioni di un documento. Utilizza Google Gemini per analizzare un riassunto della versione corrente e un riassunto di una versione precedente (recuperato da Redis) e determina se le modifiche sono "significative", restituendo 0 (simili) o 1 (molto differenti).

2.3 Storage e Versionamento

- **Redis**: Utilizzato come database vettoriale per memorizzare gli embedding dei riassunti dei file e come store chiave-valore per i metadati associati (percorso, hash del commit, riassunto testuale). La chiave per ogni voce include l'hash del commit, permettendo di indicizzare e recuperare versioni specifiche.
- **Git**: Integrato in `app.py` per creare automaticamente commit delle modifiche ai file nella directory monitorata. Questo fornisce la base per il tracciamento delle versioni. L'hash del commit Git è una parte fondamentale dell'identificatore univoco di una versione di un file in Redis.

3 Tecnologie Utilizzate

- **Python**: Per lo sviluppo del backend.
- **JavaScript (Node.js)**: Per lo sviluppo del frontend con Electron.
- **Electron**: Framework per la creazione di applicazioni desktop cross-platform con tecnologie web.
- **Redis (con Redis Stack/Search)**: Database NoSQL utilizzato per la memorizzazione di metadati e come database vettoriale per la ricerca di similarità.
- **Git**: Sistema di controllo versione distribuito, utilizzato per il versionamento dei file.
- **Google Gemini**: Modello AI generativo utilizzato per l'analisi delle query, la generazione di riassunti e il confronto tra versioni.
- **SentenceTransformers**: Libreria Python per la generazione di embedding di testo.
- **Watchdog**: Libreria Python per il monitoraggio degli eventi del file system.
- **Socket.IO / WebSockets**: Per la comunicazione real-time tra frontend e backend.