UNIVERSITÀ DEGLI STUDI DI
NAPOLI FEDERICO II

Corso di Laurea Magistrale in Ingegneria
Informatica

Dipartimento di Ingegneria Elettrica e delle
Tecnologie dell'Informazione

Progetto Big Data
Voice2Care

Candidati:

Vincenzo D'Angelo M63001595
Giorgio Di Costanzo M63001579

Professore:

Vincenzo Moscato

Anno Accademico 2024/2025

Indice

1 Introduzione

Voice2Care è un progetto che mira a semplificare la digitalizzazione dei dati clinici attraverso la trascrizione automatica di registrazioni audio e l'estrazione delle informazioni rilevanti in formato strutturato. Nasce dall'esigenza di rendere la registrazione delle informazioni rapida ed efficiente in ambienti ad alta intensità, come il pronto soccorso o il pronto intervento, dove la velocità di azione è fondamentale.
L'utilizzo di modelli avanzati di riconoscimento vocale ed elaborazione del testo consente di migliorare l'affidabilità della trascrizione, facilitandone l'interpretazione e l'utilizzo. I dati strutturati possono essere archiviati in database NoSQL, che garantiscono un accesso rapido e una gestione efficiente delle informazioni raccolte.

Nel presente elaborato proponiamo l'implementazione del sistema descritto, integrato in una dashboard interattiva che consente la visualizzazione, modifica, validazione ed esportazione dei dati estratti. L'applicazione è stata inoltre resa accessibile via internet ed è stata testata su un dataset appositamente generato.

Per garantire un'esecuzione più fluida e tempi di inferenza accettabili, è fortemente consigliato l'impiego di una GPU, in particolare durante le fasi di trascrizione ed estrazione.

Il progetto è pubblicamente disponibile nella repository GitHub [1].
Le istruzioni per l'installazione e l'utilizzo sono all'interno del file `README.md`.

1.1 Librerie e strumenti utilizzati

Il progetto è stato interamente sviluppato in Python e si basa sull'integrazione di diverse librerie, tra cui:

- openai-whisper, per il riconoscimento vocale automatico;
- transformers (HugginFace), per il caricamento e l'utilizzo di LLM
- torch, per il supporto GPU durante l'inferenza;
- librerie per la gestione e normalizzazione degli input audio;
- librerie per la manipolazione di file e dati strutturati;
- librerie per la creazione di interfacce e visualizzazioni;

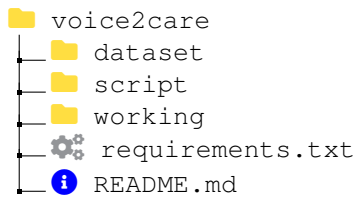
Per un elenco completo e dettagliato dei pacchetti utilizzati si rimanda al file `requirements.txt`.

Per l'elaborazione del progetto si è scelto di utilizzare la piattaforma *Kaggle*, di proprietà di Google. Essa mette a disposizione un ambiente Jupyter Notebook basato su cloud, che consente di scrivere ed eseguire codice Python direttamente dal browser.
Dopo la registrazione e la verifica tramite numero di telefono, Kaggle offre gratuitamente, per un massimo di 30 ore settimanali, le seguenti risorse computazionali:

- Hard Disk: 57,6 GB;
- CPU;
- RAM: 29 GB
- GPU: NVIDIA P100 con 16 GB di memoria dedicata

L'altro strumento utilizzato è *Ngrok*, un servizio di tunneling che consente di esporre in modo sicuro un indirizzo locale su Internet. Ngrok fornisce un link temporaneo, attraverso il quale è possibile accedere al server locale da qualsiasi dispositivo. In questo modo, è stato possibile rendere accessibile la dashboard Streamlit eseguita su Kaggle.

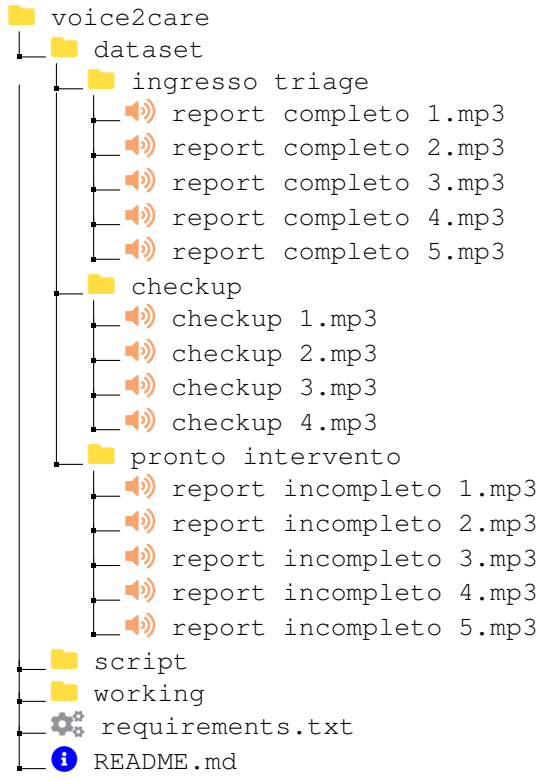
La cartella del progetto presenta la seguente struttura:



I contenuti di ciascuna cartella verranno analizzati in maniera più approfondita nelle rispettive sezioni.

1.2 Dataset

Per validare il sistema implementato, è stato generato un dataset composto da file audio contenenti informazioni cliniche simulate.



I file sono stati realizzati con l'obiettivo di riflettere scenari realistici in ambito sanitario, con le seguenti modalità:

- Registrazione vocale: alcuni file sono stati generati manualmente in condizioni controllate;
- Generazione sintetica: altri file sono stati creati utilizzando strumenti di text-to-speech basati su AI.

Le registrazioni sono state prodotte in diverse lingue per valutare le performance multilingue del sistema.

Si è deciso di considerare 3 casi d'uso distinti:

- Ingresso triage: l'audio simula il report prodotto all'arrivo di un paziente in pronto soccorso. Il modulo associato presenta un numero fisso di campi, tutti obbligatori. Pertanto, si prevede la generazione di un testo strutturato completo, contenente 26 campi compilati.

- Check-up giornaliero: l'audio simula il report di un controllo di routine effettuato quotidianamente sul paziente. Anche in questo caso, il modulo associato presenta un numero fisso di campi, tutti obbligatori. Pertanto, si prevede la generazione di un testo strutturato completo, contenente 12 campi compilati.

- Pronto intervento: l'audio simula il report prodotto da un'unità di pronto intervento. Il modulo associato presenta un numero fisso di campi, ma non tutti sono obbligatori. Pertanto, si prevede la generazione di un testo strutturato incompleto, contenente un numero variabile di campi compilati.

Ai dati estratti dal modello di linguaggio vengono aggiunti automaticamente ulteriori campi, quali ad esempio l'ora di ingresso e la tipologia di documento. L'interfaccia utente e la struttura del documento esportato sono stati adattati e personalizzati in base alle specifiche esigenze di ciascun caso d'uso.
Tutti i file sono stati salvati in formato `.mp3` e organizzati all'interno della cartella `\dataset`, suddivisi a seconda del caso d'uso di appartenenza.

2 Metodologia

In questo capitolo descriviamo l'architettura, la metodologia e le tecnologie adottate per la realizzazione degli obiettivi presentati nel capitolo precedente. L'architettura proposta è la seguente:

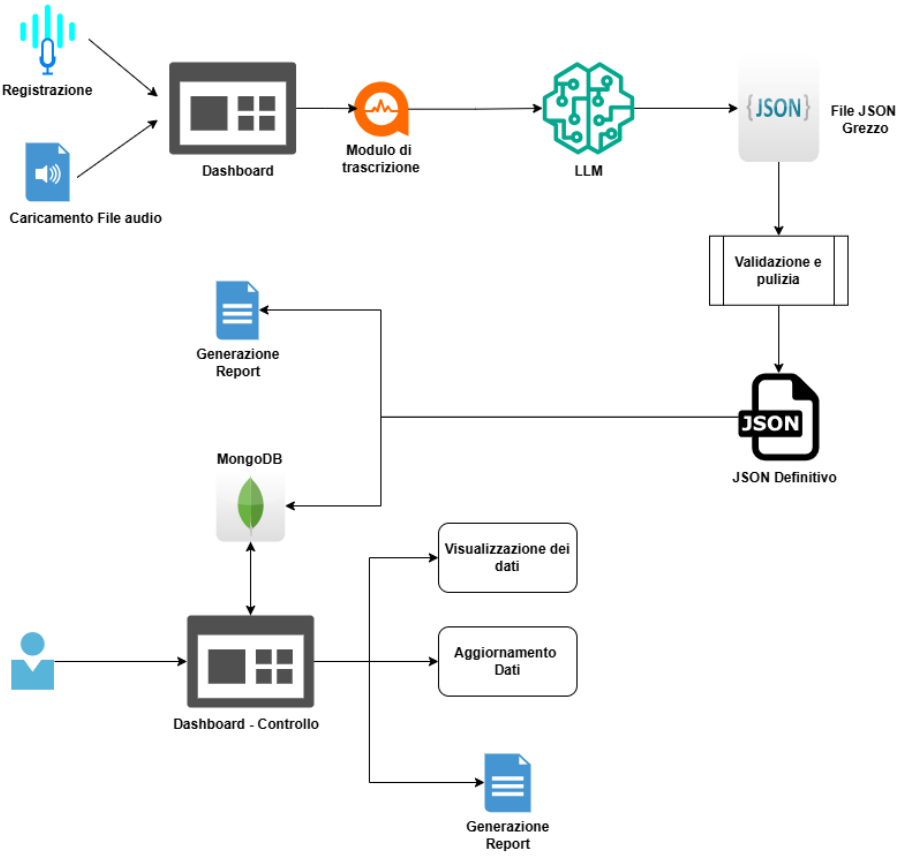
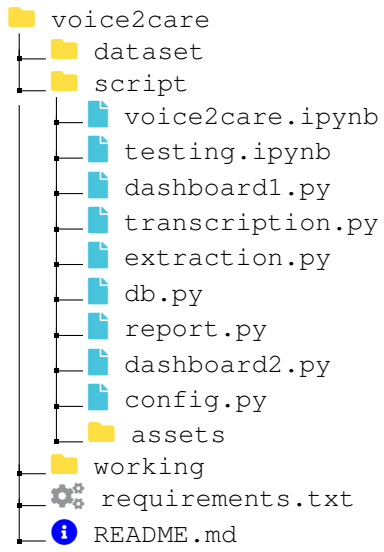


Figura 1: Architettura

L'utente può scegliere se registrare un audio direttamente tramite l'applicazione oppure caricare un file audio precedentemente registrato. L'audio viene fornito come input a un modello di riconoscimento vocale, che ne produce una trascrizione testuale. Tale trascrizione viene successivamente inserita, insieme ai comandi e i campi da estrarre, nel prompt di un LLM. Dalla risposta del modello si estrae un oggetto JSON, che viene sottoposto a operazioni di pulizia e validazione. Infine, l'utente ha la possibilità di esportare o scaricare il report generato.
La seconda interfaccia consente di visualizzare, ricercare, modificare e salvare i dati precedentemente esportati.

I paragrafi che seguono presentano un'analisi dettagliata dei componenti dell'architettura, implementati all'interno della sottocartella `\script`.



2.1 Dashboard principale

La dashboard principale è implementata nel file `dashboard1.py` e consente la generazione di un report strutturato a partire da un file audio. Nella sidebar, l'utente può scegliere se registrare un nuovo audio o caricarne uno già esistente, e selezionare la modalità di utilizzo: check-up, report completo o report incompleto.
Una volta definito l'audio, vengono attivati i moduli di trascrizione e di estrazione. In caso di errore durante l'estrazione del file JSON, il sistema tenta automaticamente una seconda esecuzione del modulo di estrazione.
Se anche il secondo tentativo fallisce, l'esecuzione viene interrotta e l'utente riceve un messaggio di errore che lo invita a registrare nuovamente l'audio.

Quando l'estrazione del JSON va a buon fine, la visualizzazione risultante è la seguente: vengono visualizzati a schermo la trascrizione e i vari campi estratti. Sotto ciascun campo può comparire un messaggio di errore che segnala una possibile anomalia, come dati incongruenti o potenziali allucinazioni generate dal modello LLM. Questo avviene grazie alla funzione di validazione automatica integrata nel sistema, che verifica la coerenza e la correttezza dei dati estratti.

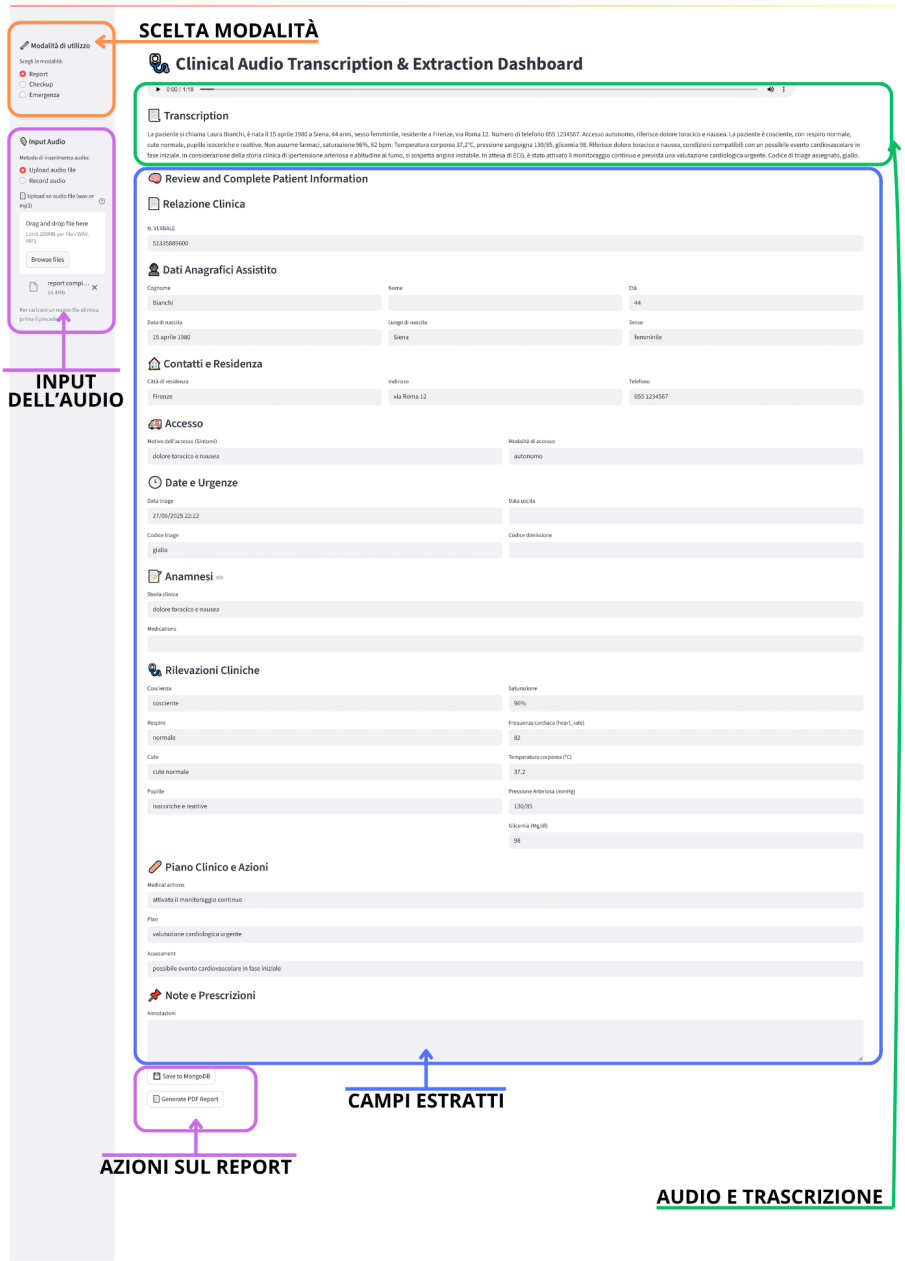


Figura 2: Dashboard 1

La dashboard è stata sviluppata utilizzando il framework open-source *Streamlit*.

2.2 Modulo di trascrizione

Il file `transcription.py` implementa il modulo di trascrizione, che riceve in ingresso un file audio e restituisce in uscita il testo trascritto, che viene successivamente rielaborato per gestire simboli o abbreviazioni.
Per la trascrizione è stato utilizzato il modello di riconoscimento vocale automatico Whisper [2], sviluppato da OpenAI. È stato addestrato su un ampio dataset di audio multilingua e multitask raccolti dal web, il che lo rende particolarmente robusto al rumore e in grado di riconoscere numerose lingue.
Il modello è disponibile in diverse dimensioni, ciascuna con un compromesso tra velocità di esecuzione, occupazione in memoria e accuratezza.
Whisper non necessita della preimpostazione della lingua e supporta anche la traduzione automatica verso l'inglese.

Per l'implementazione del sistema è stata scelta la variante *Medium*, in quanto si è dimostrata più affidabile delle versioni più leggere nella trascrizione dei report in lingua italiana, e significativamente più veloce rispetto alla variante *Large*. Questa versione presenta circa 769 milioni di parametri e ha un peso complessivo di circa 1,5 GB, rappresentando un buon compromesso tra accuratezza e prestazioni.

2.3 LLM

All'interno del file `extraction.py` è stato sviluppato il modulo che si occupa dell'estrazione e della formattazione in formato JSON delle informazioni contenute nel testo non strutturato.
Per questo specifico task si è deciso di utilizzare un *Large Language Model (LLM)*, in virtù della sua capacità di estrarre informazioni rilevanti, comprendere il testo e adattarsi in modo flessibile a diverse strutture e contesti linguistici.

Per estrarre le informazioni contenute nella trascrizione, il modello viene interrogato tramite un prompt fisso, strutturato secondo uno schema preciso e composto da quattro sezioni principali:

1. Istruzione iniziale: una richiesta esplicita al modello di identificare ed estrarre le informazioni rilevanti dal testo, organizzandole in formato JSON;
2. Testo della trascrizione: il contentuo testuale ottenuto dalla fase di trascrizione;
3. Vincoli e requisiti per il file JSON:
    - i campi e i relativi valori devono essere espressi nella stessa lingua dell'input;
    - il file JSON deve essere compatto e ben strutturato;
    - deve essere restituita una stringa vuota in corrispondenza di ogni campo non presente nella trascrizione.

4. Elenco dei campi richiesti nel JSON: include informazioni anagrafiche, parametri vitali, stato clinico, anamnesi, sintomi, interventi e valutazioni.

Dopo aver ricevuto il prompt in input, il modello restituisce una risposta contentente l'oggetto JSON già formattato secondo le specifiche fornite.
Il modello selezionato per questo compito appartiene alla famiglia Gemma, una serie di modelli text-to-text open-source sviluppati da Google. Supportano oltre 140 lingue e sono progettati per risolvere diverse tipologie di task, come question answering, text summarization ed elaborazione di prompt complessi.
Una caratteristica chiave di questi modelli è il basso consumo di risorse computazionali, che li rende adatti all'esecuzione anche su hardware non particolarmente performanti.

Nello specifico si è deciso di utilizzare il modello gemma-2-2b-it [3], che conta 2 miliardi di parametri. Pur essendo più leggero rispetto ai modelli più grandi, garantisce prestazioni adeguate per al compito assegnato, grazie all'addestramento su un ampio dataset. Inoltre, essendo *instruction-tuned*, rispetto alla versione base gemma-2-2b, è in grado di seguire con maggiore precisione le istruzioni fornite, risultando più accurato e meno soggetto ad allucinazioni.

2.3.1 Estrazione, validazione e pulizia del file JSON

Una volta ottenuta la risposta dal modello contenente il JSON, quest'ultimo viene estratto tramite parsing basato sul riconoscimento delle parentesi graffe. Successivamente, il JSON estratto viene passato a una funzione di pulizia che opera sui campi relativi alle misurazioni cliniche, che includono parametri fondamentali per la valutazione dello stato di salute del paziente. Questi valori sono spesso soggetti a variazioni nel formato o alla presenza di simboli non uniformi. Per questo motivo, necessitano di un'accurata normalizzazione per garantire coerenza e affidabilità nei dati trattati. La funzione implementata agisce sui seguenti campi:

- Ossigenazione: estrazione del valore numerico e aggiunta automatica del simbolo "%"se assente;
- Pressione sanguigna: individuazione dei due valori numerici separati e inserimento del separatore"/"in caso di mancanza;
- Temperatura corporea, battito cardiaco e glicemia: eliminazione di caratteri non numerici.

Dopo l'esecuzione della funzione di pulizia, viene applicata una funzione di validazione, che analizza i diversi campi per rilevare eventuali dati incongruenti o allucinazioni, ovvero informazioni non presenti o non coerenti con il contenuto della trascrizione. In particolare, vengono effettuati controlli su:

- Nome, cognome e città: devono avere più di due caratteri ed essere presenti nella trascrizione;
- Indirizzo: deve contenere più di cinque caratteri ed essere citato nel testo;
- Numero di telefono: deve essere costituito da solo cifre intere, avere almeno sette cifre ed essere presente nella trascrizione;
- Età: deve essere un numero intero compreso tra 0 e 130 menzionato nel testo;
- Anno di nascita: viene derivato dalla data di nascita. Deve risultare nel testo ed essere coerente con l'età;
- Temperatura corporea: deve essere un valore numerico compreso tra 0 e 50°C e citato nella trascrizione;
- Ossigenazione: deve contenere un valore numerico inferiore a 100 presente nel testo;
- Frequenza cardiaca: deve essere un numero compreso tra 0 e 700 e corrispondere a quanto dichiarato nella trascrizione;
- Glicemia: deve essere un valore numerico compreso tra 20 e 500 e presente nella trascrizione.

Al termine della validazione, la funzione restituisce un elenco degli eventuali errori riscontrati. In caso di anomalie, nella dashboard viene visualizzata un'icona di avviso sotto il campo corrispondente, in modo che l'operatore possa decidere di correggere manualmente il valore.

2.4 MongoDB

Il file `db.py` contiene la logica per la memorizzazione e il salvataggio di tutti i report generati. Per questo motivo, rappresenta il nodo centrale che collega le due dashboard.

Le funzioni implementate per interagire con il database sono le seguenti:

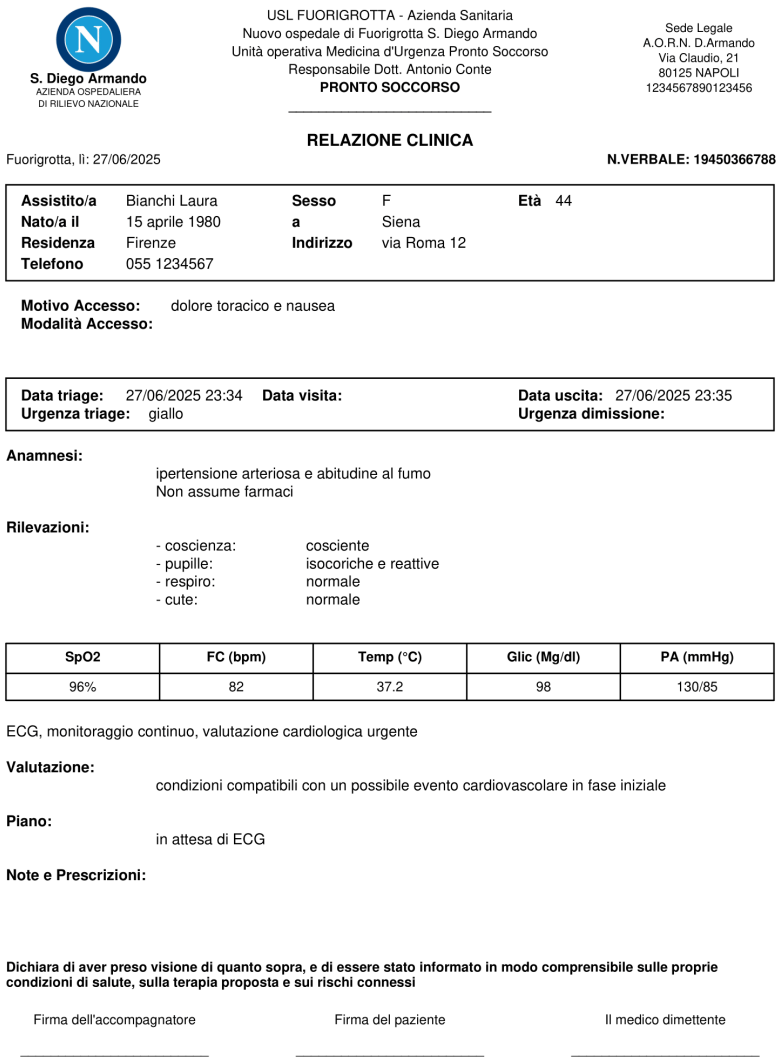
- Salvataggio del report (`save_to_mongo`): salva un oggetto JSON nel database; se un documento con lo stesso identificativo è già presente, viene sovrascritto;
- Recupero di tutti i documenti (`get_all_documents`): restituisce tutti i documenti presenti nella collezione;
- Caricamento di un documento (`load_from_mongo`): dato in input un ID, restituisce il documento JSON corrispondente, se presente;
- Caricamento dell'ultimo documento (`get_latest_document`): restituisce l'ultimo documento caricato.

Il database utilizzato è MongoDB [4]: un database NoSQL che immagazzina dati in formati diversi dalle tradizionali tabelle e non richiede uno schema predefinito, garantendo così maggiore flessibilità e scalabilità. È un database document-oriented open-source, che utilizza *Binary JSON (BSON)* per la memorizzazione dei dati.
Supporta dati eterogenei grazie a uno schema flessibile che non vincola la struttura dei documenti e consente di definire indici sui singoli campi o su combinazioni di campi per ottimizzare le query. Inoltre, permette la scalabilità orizzontale tramite sharding e garantisce la sicurezza end-to-end dei dati. Ogni documento è autosufficiente e rende possibili tutte le operazioni CRUD.

2.5 Generazione report

All'interno del file `report.py` è stato sviluppato il modulo incaricato della generazione dei report in formato PDF. Per questo scopo, è stata utilizzata la libreria *ReportLab*, che consente un controllo preciso sulla formattazione e disposizione degli elementi.

Il report viene stampato secondo due schemi predefiniti, corrispondenti rispettivamente a un report ospedaliero completo o a un check-up giornaliero.



S. Diego Armando
AZIENDA OSPEDALIERA
DI RILIEVO NAZIONALE

USL FUORIGROTTA - Azienda Sanitaria
Nuovo ospedale di Fuorigrotta S. Diego Armando
Unità operativa Medicina d'Urgenza Pronto Soccorso
Responsabile Dott. Antonio Conte
PRONTO SOCCORSO

Sede Legale
A.O.R.N. D.Armando
Via Claudio, 21
80125 NAPOLI
1234567890123456

RELAZIONE CLINICA

Fuorigrotta, lì: 27/06/2025 **N.VERBALE: 19450366788**

Assistito/a	Bianchi Laura	**Sesso**	F	**Età**	44
Nato/a il	15 aprile 1980	**a**	Siena		
Residenza	Firenze	**Indirizzo**	via Roma 12		
Telefono	055 1234567				

Motivo Accesso: dolore toracico e nausea
Modalità Accesso:

Data triage:	27/06/2025 23:34	**Data visita:**		**Data uscita:**	27/06/2025 23:35
Urgenza triage:	giallo			**Urgenza dimissione:**	

Anamnesi:
ipertensione arteriosa e abitudine al fumo
Non assume farmaci

Rilevazioni:
- coscienza: cosciente
- pupille: isocoriche e reattive
- respiro: normale
- cute: normale

SpO2	FC (bpm)	Temp (°C)	Glic (Mg/dl)	PA (mmHg)
96%	82	37.2	98	130/85

ECG, monitoraggio continuo, valutazione cardiologica urgente

Valutazione:
condizioni compatibili con un possibile evento cardiovascolare in fase iniziale

Piano:
in attesa di ECG

Note e Prescrizioni:

Dichiara di aver preso visione di quanto sopra, e di essere stato informato in modo comprensibile sulle proprie condizioni di salute, sulla terapia proposta e sui rischi connessi

Firma dell'accompagnatore _____ Firma del paziente _____ Il medico dimettente _____

Figura 3: Report

Questa funzionalità è accessibile in entrambe le dashboard: nella prima subito dopo la creazione del report, nella seconda in qualsiasi momento per consultare e ristampare un documento salvato. In questo modo, l'operatore sanitario può ottenere rapidamente un documento stampabile da allegare alla cartella clinica del paziente.

2.6 Dashboard 2

La seconda dashboard, implementata nel file `dashboard2.py`, è progettata per il personale medico delle strutture sanitarie, con l'obiettivo di fornire un accesso rapido ed efficiente alle informazioni contenute nei report generati.

Le funzionalità principali includono:

- Accesso immediato ai report attraverso la selezione di specifici campi;
- Utilizzo di una sidebar per ricerca avanzata, con filtri per categoria o per singolo campo;
- Aggiornamento in tempo reale dei dati, al fine di garantire una gestione tempestiva delle emergenze;
- Evidenziazione automatica dei casi urgenti, visualizzati in rosso per una rapida identificazione;
- Visualizzazione della trascrizione audio associata a ciascun report;
- Possibilità di modificare e aggiornare i dati di ogni singolo report;
- Generazione e stampa del report in formato PDF.

L'interfaccia è suddivisa in diverse componenti, ognuna con una funzione specifica.

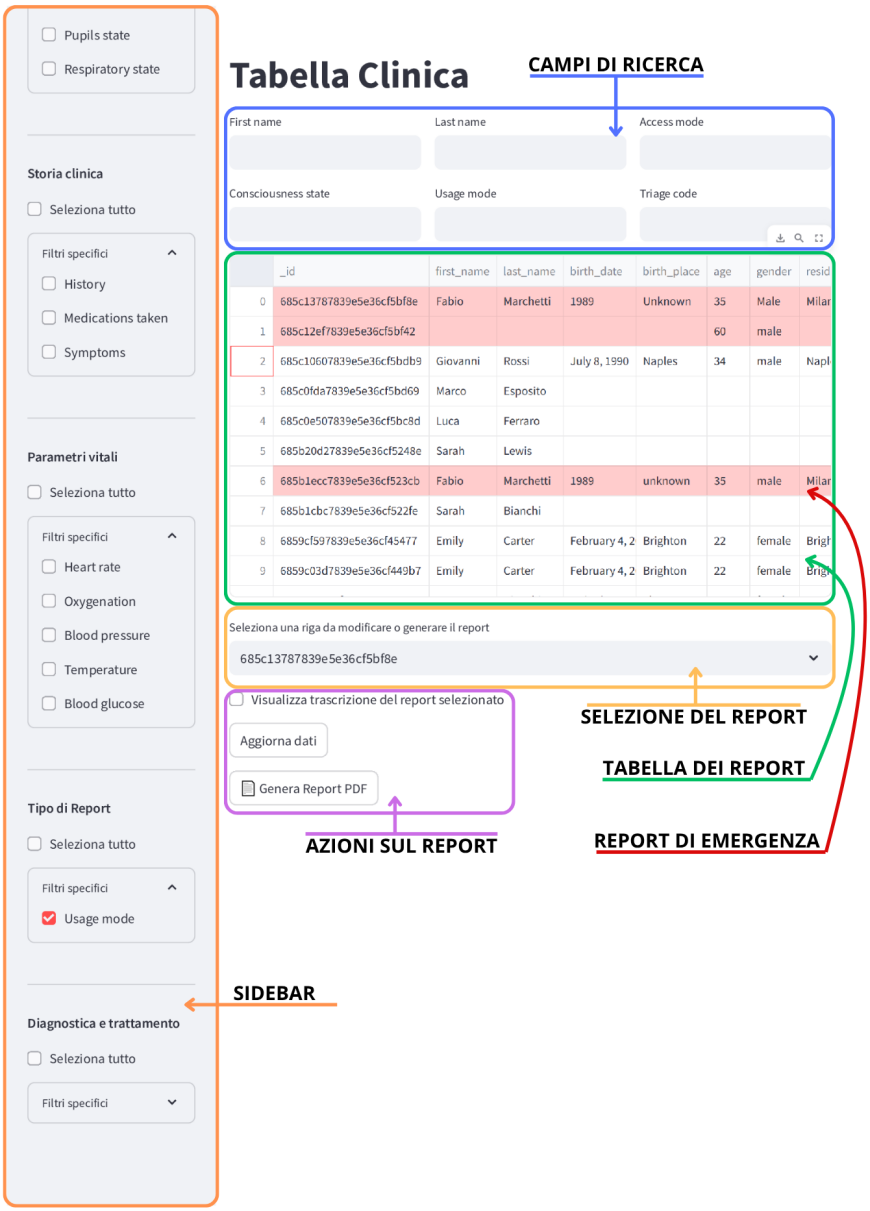


Figura 4: Dashboard 2

Nella parte superiore sono presenti i campi di ricerca predefiniti: *nome*, *cognome*, *codice di triage*, *tipologia di report*, *modalità di accesso* e *stato di coscienza*. Sul lato sinistro è disponibile una sidebar interattiva, che permette di selezionare o deselezionare i campi di ricerca, sia per categoria che singolarmente. Le categorie e i relativi campi sono:

- Anagrafica: nome, cognome, data di nascita, luogo di nascita, età, genere, città di residenza, indirizzo di residenza, telefono;
- Accesso: modalità di accesso;
- Condizioni: stato della pelle, stato di coscienza, stato delle pupille, stato della respirazione;
- Storia clinica: anamnesi, medicazioni effettuate, sintomi;
- Parametri vitali: battito cardiaco, ossigenazione, pressione sanguigna, temperatura corporea, glicemia;
- Tipo di report: modalità di uso;
- Diagnostica e trattamento: test diagnostici, valutazione, piani terapeutici, codice triage.

Centralmente è presente una tabella con l'elenco dei report; le righe evidenziate in rosso corrispondono alle emergenze, così da attirare immediatamente l'attenzione dell'operatore.

Sotto la tabella è possibile selezionare un report per visualizzare la trascrizione dell'audio da cui è stato generato, aggiornare i dati o generare il report in formato PDF.

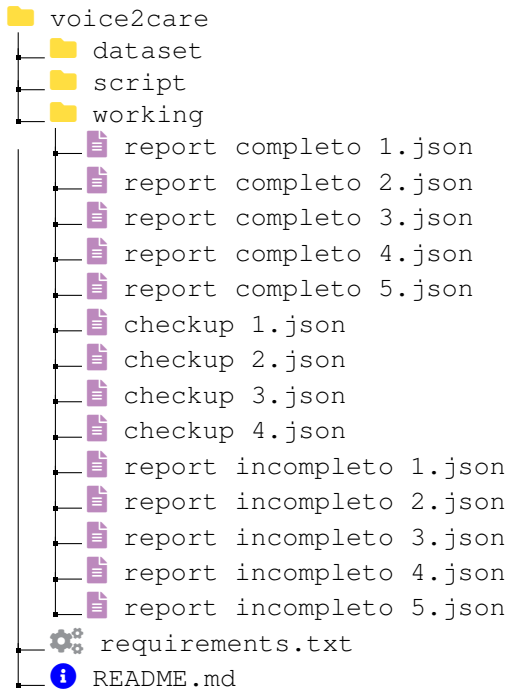
Nel caso si desideri aggiornare i dati, l'interfaccia mostra un menu a tendina contenente le varie categorie, ciascuna con i relativi sottocampi. Una volta completate le modifiche, il report viene aggiornato automaticamente nel database.

Anche la seconda dashboard è stata realizzata con Streamlit.
Le selezioni e lo stato dell'utente sono mantenuti tramite variabili `session_state`, garantendo coerenza dell'interfaccia tra le diverse azioni.
La tabella si aggiorna in tempo reale: ogni 5 secondi viene interrogato il database per verificare a presenza di nuovi documenti e, in caso di cambiamenti, la tabella viene aggiornata automaticamente.
La tabella è stata implementata con l'aiuto del componente *fragment*, che consente di aggiornare solo la tabella senza ricaricare l'intera dashboard, permettendo un'interazione fluida e senza blocchi.

La combinazione di queste funzionalità consente di monitorare in modo rapido ed efficace le condizioni cliniche dei pazienti.

3 Risultati ottenuti

La fase di testing è stata condotta sul dataset descritto precedentemente, utilizzando il notebook `testing.ipynb`. Ogni file JSON ottenuto è stato valutato manualmente, adottando criteri di verifica flessibili ma coerenti. I risultati dei test sono consultabili all'interno della cartella `\working`.



In tutti i casi il sistema è stato in grado di identificare correttamente i parametri richiesti, con rare eccezioni. Le imprecisioni riscontrate hanno riguardato principalmente il campo relativo alla modalità di accesso e alcuni errori riconducibili alla fase di trascrizione automatica.

Per migliorare l'accuratezza dei risultati, la richiesta di estrazione del campo `access_mode` è stata resa più chiara e dettagliata. Inoltre, è stato introdotto un secondo prompt dedicato alla correzione degli errori di trascrizione:

```
prompt = f"""
Rewrite the following clinical text by correcting any transcription errors.
   ----
   Requirements:
   - Translate corrected text into the same language as the clinical text.

Clinical text:
{text}

Corrected text:
"""
```

L'integrazione di queste modifiche ha permesso di risolvere le criticità riscontrate, migliorando la qualità dell'output finale.

4 Conclusioni e lavori futuri

Il progetto proposto ha confermato l'efficacia e la fattibilità di un sistema automatico per la trascrizione e l'estrazione strutturata di dati clinici a partire da registrazioni audio, con applicazioni particolarmente rilevanti in contesti ad alta intensità come pronto soccorso e pronto intervento. L'integrazione di modelli avanzati di riconoscimento vocale e linguaggio naturale ha reso possibile un'elaborazione efficiente delle informazioni cliniche, potenzialmente in grado di supportare concretamente il personale sanitario nella redazione di report digitali accurati.

Nonostante i risultati promettenti, il sistema presenta margini di miglioramento, in particolar modo per quanto riguarda la qualità della trascrizione in ambienti reali. È infatti fondamentale assicurare prestazioni affidabili anche in presenza di rumori di fondo, bassa qualità audio o pronunce imprecise. In questa prospettiva, uno sviluppo particolarmente rilevante potrebbe riguardare l'implementazione di un correttore automatico specializzato, addestrato su coppie di dati trascritti e corretti, in grado di adattarsi al linguaggio specifico del dominio medico.

Ulteriori sviluppi potrebbero includere l'estensione del sistema ad una più ampia varietà di casi d'uso clinici, l'esportazione automatica dei dati verso cartelle cliniche elettroniche e l'adozione di database neurali. Quest'ultima soluzione, in particolare, consentirebbe di potenziare le capacità analitiche e di ricerca sul corpus clinico, facilitando interrogazioni semantiche complesse.

Riferimenti

[1] Giorgio Di Costanzo, Vincenzo D'Angelo. *Voice2Care - GitHub repository*. `https://github.com/vincenzodan/voice2care`. 2025.

[2] OpenAI. *Whisper Medium*. Automatic Speech Recognition Model. 2022. URL: `https://github.com/openai/whisper`.

[3] Google. *Gemma 2 2B IT*. Large Language Model. 2024. URL: `https://huggingface.co/google/gemma-2-2b-it`.

[4] *MongoDB*. `https://www.mongodb.com`. MongoDB, Inc., 2024.

[5] *Materiale del corso di Big Data*. Universtà degli Studi di Napoli Federico II, 2025.