Nel Db che tipo di file posso salvare? Solo txt, o anche gli Html?

**ChatGPT ha detto:**

Nel database SQLite (come crisp\_projects.db), puoi salvare **qualsiasi tipo di contenuto**, non solo .txt. Quello che importa non è l’estensione, ma **come il contenuto viene trattato e memorizzato**.

Ecco cosa puoi salvare:

**✅ 1. Testo semplice (es. .txt, .json)**

* Tipo campo: TEXT
* Esempio: contenuti di prompt, risposte, metadata, JSON serializzati.

**✅ 2. HTML**

* Tipo campo: TEXT
* Salvi l’intero page\_source come stringa HTML.
* Puoi poi ricaricarlo, visualizzarlo in un iframe, o analizzarlo con BeautifulSoup o simili.

**✅ 3. File binari (es. immagini, PDF)**

* Tipo campo: BLOB (Binary Large Object)
* Puoi salvare qualunque tipo di file, anche .jpg, .pdf, .zip, ecc.
* Ma è più complesso da gestire: richiede conversione read(file, 'rb') → BLOB, e il recupero deve essere decodificato correttamente.

**✳️ Best practice per HTML**

Salva il contenuto in:

python

CopiaModifica

cursor.execute("INSERT INTO pagine\_html (project\_id, html\_content) VALUES (?, ?)", (project\_id, html\_code))

Con html\_code tipo stringa (TEXT). Così lo puoi facilmente rileggere o ispezionare.

**Vuoi che ti mostri come creare una tabella per salvare HTML completi nel tuo crisp\_projects.db?**

**Hai detto:**

Allora siccome ho già le cartelle: C:\Users\a\Documents\PubliScript\_Refactored\backups C:\Users\a\Documents\PubliScript\_Refactored\output\genspark\_formatted Il DB non si può aggiornare da queste cartelle ogni volta che arriva un nuovo file?

**ChatGPT ha detto:**

Sì, **puoi aggiornare automaticamente il database SQLite (crisp\_projects.db) ogni volta che arriva un nuovo file** in una di quelle cartelle (backups o output/genspark\_formatted) — ma serve uno **script Python di sincronizzazione** che faccia da ponte tra file system e database.

**✅ Cosa deve fare lo script**

Ecco la logica:

1. **Scandisce le cartelle backups/ e output/genspark\_formatted/**.
2. **Identifica nuovi file** non ancora presenti nel DB.
3. **Estrae dati chiave** (es. nome file, timestamp, keyword da filename).
4. **Li inserisce nel database** (projects, project\_variables, ecc.).
5. **Evita duplicati**, ad esempio usando:
   * hash dei contenuti,
   * nome file univoco,
   * ID progetto già presenti nel DB.

**🔧 Esempio: sincronizzazione file HTML da output/genspark\_formatted**

python

CopiaModifica

import os

import sqlite3

from datetime import datetime

DB\_PATH = "crisp\_projects.db"

HTML\_FOLDER = "output/genspark\_formatted"

def sync\_html\_files\_to\_db():

conn = sqlite3.connect(DB\_PATH)

cursor = conn.cursor()

# Assicurati che esista la tabella

cursor.execute("""

CREATE TABLE IF NOT EXISTS html\_snapshots (

id INTEGER PRIMARY KEY AUTOINCREMENT,

filename TEXT UNIQUE,

content TEXT,

created\_at TEXT

)

""")

for fname in os.listdir(HTML\_FOLDER):

if fname.endswith(".html"):

full\_path = os.path.join(HTML\_FOLDER, fname)

with open(full\_path, "r", encoding="utf-8") as f:

content = f.read()

try:

cursor.execute("""

INSERT INTO html\_snapshots (filename, content, created\_at)

VALUES (?, ?, ?)

""", (fname, content, datetime.now().isoformat()))

print(f"✅ Inserito: {fname}")

except sqlite3.IntegrityError:

print(f"⚠️ Già presente: {fname}")

conn.commit()

conn.close()

**📦 Estendibile anche a:**

* file .txt da backups/,
* estrazione keyword dal nome file (es. context\_GMAT\_2025... → GMAT),
* aggiornamento tabella projects, project\_variables, ecc.

**Vuoi che ti prepari uno script completo che:**

* sincronizzi entrambe le cartelle (backups/ e genspark\_formatted/)
* aggiorni il database coerentemente?

Posso fartelo in 1 blocco.

**Hai detto:**

Per esempio ora come funziona? Come si salva su DB attualmente?

**ChatGPT ha detto:**

Attualmente, **il salvataggio su database in PubliScript avviene in modo automatico ogni volta che un’analisi viene creata o aggiornata**, tramite il modulo DatabaseManager, in particolare usando:

**🔁 Struttura attuale di salvataggio su DB**

Quando l’analisi viene generata o continua:

1. **Si crea un progetto nella tabella projects**
   * Contiene: id, name, creation\_date, last\_updated.
2. **Si salvano le variabili nella tabella project\_variables**
   * Es. KEYWORD, TITOLO\_LIBRO, VOICE\_STYLE, ecc.
3. **Si salvano le risposte passo-passo nella tabella project\_results**
   * Campi: project\_id, prompt\_id (es. "CP-2"), result\_text, timestamp.
4. Se attiva la modalità incrementale, si usano anche:
   * incremental\_responses (sezione e risposta per step incrementali lunghi).

**🧠 Codice rilevante**

Nel tuo DatabaseManager, il salvataggio avviene per esempio da:

* self.db\_manager.save\_analysis\_step(project\_id, step\_id, result\_text)
* o metodi simili chiamati da chat\_manager o book\_builder.

**📁 E quindi perché non salva i file .html o .txt da cartella?**

Perché:

* Tutto il salvataggio è **gestito runtime dal codice**, non c’è **nessuna funzione che scandisca le cartelle** backups/ o genspark\_formatted/.
* Le **versioni .txt o .html** sono *solo backup locali* o file di debug/manuali.
* Il DB salva **solo i dati strutturati**, non i contenuti dei file locali.

**✅ Cosa manca ora**

Per salvare i file .html o .txt nel database serve:

* una **funzione esplicita** che legga quelle cartelle;
* una logica che **crei un progetto nel DB** se manca;
* e che **salvi i contenuti (txt/html)** come testo nella tabella project\_results o in una tabella apposita (es. snapshots, imports, etc.).

Vuoi che ti prepari una **funzione importa\_backup\_da\_file()** per inserire un file .txt di backup o .html di Genspark nel database come nuovo progetto o aggiornamento di uno esistente?

**Hai detto:**

Aspè non voglio fare cagate, se la logica attuale è migliore della mia teniamola, che dici? Oppure no, non so, non funziona una mazza ma se si fa così, allora va bene.

**ChatGPT ha detto:**

Hai ragione a farti questa domanda, e ti rispondo con la massima chiarezza.

**🧠 La logica attuale è robusta e strutturata per un flusso guidato.**

È fatta per:

* tracciare *ogni progetto* come entità univoca nel database;
* salvare lo **stato delle analisi** (fasi CRISP, risposte, keyword, ecc.);
* ricostruire un progetto in qualsiasi momento via interfaccia.

Questo è fondamentale per software seri, modulari, scalabili.

**❌ Però ha un punto debole: i file backups/ e output/genspark\_formatted/ sono fuori sincronizzazione dal DB.**

* Quelli .txt in backups/ servono **solo come copia di sicurezza**, non sono più letti una volta salvati.
* Gli .html di Genspark sono **file di debug/manuale**: puoi guardarli, ma **non sono mai salvati nel DB** né legati a un progetto.

Quindi:

**La logica attuale è corretta**, ma manca un **collegamento tra i file locali e il database**. È come se avessi due mondi che non si parlano.

**✅ Soluzione migliore (mia raccomandazione):**

1. **Manteniamo il sistema attuale di database com'è** (è solido).
2. **Aggiungiamo un modulo opzionale** che:
   * legge i file .txt o .html da backups/ e output/;
   * ti chiede se vuoi **importarli come nuovi progetti** nel database;
   * li associa a una keyword e li rende visibili nell’interfaccia.

Così:

* non perdi nulla;
* resti coerente con l'architettura esistente;
* puoi caricare vecchi lavori se vuoi, ma solo quando serve.

Vuoi che ti prepari un primo script import\_file\_into\_db.py per testarlo sui tuoi backup attuali?

**Hai detto:**

In pratica, per capire, ora nel DB ci sono solo txt tipo quelli del context? Tipo o identici?

**ChatGPT ha detto:**

Esatto, *nel database* non vengono salvati né HTML né PDF, ma solo **dati testuali strutturati**, cioè:

**✅ Cosa contiene il database:**

1. **Variabili testuali** (project\_variables), come:
   * KEYWORD
   * TITOLO\_LIBRO
   * VOICE\_STYLE
   * CONTENT\_PILLARS
   * ecc.
2. **Risultati delle fasi** (project\_results), salvati come testo (result\_text) associato a uno step\_id tipo "CS-1", "CP-2", ecc.
3. **Contesto iniziale** (context.txt)  
   Viene copiato (non identico, ma molto simile) e salvato come valore in project\_variables, sotto nomi tipo CONTEXT\_ORIGINALE, CONTENT\_PILLARS, ecc.

**❌ Cosa NON viene salvato nel DB:**

* File .html (es. dump Genspark)
* File .txt di backup **non legati al progetto**
* File .json di esportazione manuale
* File .md, .pdf, immagini, ecc.

**In sintesi**

Nel DB c’è solo il **cuore informativo testuale** del progetto, non i file fisici.

Vuoi che ti mostri un esempio concreto tra un backup .txt e ciò che viene salvato nel database per confronto?

**Hai detto:**

Si, perché non ho capito una mazza di che cazzo ci sta

**ChatGPT ha detto:**

Perfetto. Facciamo chiarezza *con un esempio reale*, confrontando:

**📁 1. File fisico context\_Living\_Trust\_20250501\_150429.txt (backup)**

Contiene probabilmente qualcosa del genere:

vbnet

CopiaModifica

KEYWORD: Living Trust

VOICE\_STYLE: Competente ma accessibile

TITOLO\_LIBRO: Living Trust 2025: A Simple Guide for Peace of Mind

CONTENT\_PILLARS:

- Cos'è un living trust

- Benefici principali

- Come si crea

- Errori comuni da evitare

...

Questo è un **file .txt completo**, scritto in chiaro. Sta nella cartella backups.

**🗃️ 2. Contenuto salvato nel database (SQLite)**

Dentro la tabella project\_variables, troviamo:

| **project\_id** | **name** | **value** |
| --- | --- | --- |
| PROJ\_20250501\_150429 | KEYWORD | Living Trust |
| PROJ\_20250501\_150429 | VOICE\_STYLE | Competente ma accessibile |
| PROJ\_20250501\_150429 | TITOLO\_LIBRO | Living Trust 2025: A Simple Guide ... |
| PROJ\_20250501\_150429 | CONTENT\_PILLARS | - Cos'è un living trust\n- Benefici ... |

E nella tabella project\_results, trovi anche:

| **project\_id** | **step\_id** | **result\_text** |
| --- | --- | --- |
| PROJ\_20250501\_150429 | CS-1 | Analisi keyword Living Trust... |
| PROJ\_20250501\_150429 | CP-1 | Buyer Persona per Living Trust... |
| ... | ... | ... |

**🔍 Differenze chiave**

| **File .txt** | **Database project\_variables & results** |
| --- | --- |
| È un **salvataggio completo** | È una **struttura dati modulare** |
| Utile per backup o ripristino | Utile per analisi, dashboard, recupero |
| Leggibile a colpo d’occhio | Accessibile solo via codice/GUI |
| Nessun legame interno al progetto DB | Collegato tramite project\_id |

**✅ Conclusione**

* Il **file backup .txt** è una copia diretta dello stato del contesto o di fasi.
* Il **database** spezzetta e struttura quei contenuti per permettere ripristino, visualizzazione, e avanzamento analisi.

Vuoi che ti mostri *esattamente* cosa contiene il DB per un progetto specifico tipo "Living Trust"?

**Hai detto:**

Ok, comincio a arburare e le rotelle nella testa iniziano a girare. Quindi ora ti chiedo ma a che serve sta cacata di DB fatto in questo modo?

**ChatGPT ha detto:**

Ottima domanda. Te lo dico senza mezzi termini: **il database così com'è serve a gestire *meglio* lo stato e il recupero delle analisi**, ma non è perfetto né completo. Vediamo a cosa serve davvero… e a cosa no.

**✅ PERCHÉ ESISTE IL DB (vantaggi)**

| **Funzione del DB** | **Perché è utile** |
| --- | --- |
| 🔁 **Riprendere da dove avevi lasciato** | Sa qual è l’ultima fase completata (es: CS-2), così non riparti da zero. |
| 📊 **Gestire più progetti separati** | Ogni progetto ha un project\_id, può avere keyword, titolo, stile voce, risultati, ecc. |
| 🧠 **Mostrare “Carica analisi esistente”** | Il dropdown in interfaccia legge dal DB le analisi con nome/keyword/data. |
| 💾 **Archiviazione strutturata** | Separare dati, prompt, risposte per analisi avanzate (es: quante fasi completate?). |
| 🧪 **Debug, test, export JSON** | Le funzioni esportazione/dettaglio leggono tutto da lì. |

**❌ COSA NON FA (o fa male)**

| **Limiti del DB attuale** | **Effetto collaterale** |
| --- | --- |
| 😵‍💫 **Non sincronizza i file .txt o .html** | Se fai una cosa a mano fuori dal sistema, il DB non se ne accorge. |
| 🕳️ **Manca una cronologia completa delle modifiche** | Se modifichi un file context e lo salvi di nuovo, il DB non tiene traccia. |
| 🐌 **Non sempre aggiornato in tempo reale** | Se c'è un errore nell’interfaccia, il DB può non avere tutto. |
| 🤐 **Contenuti non leggibili subito** | Non vedi a colpo d’occhio il contesto o la risposta come in un .txt. |
| 🔧 **Serve codice per fare tutto** | Non puoi semplicemente “aprire e modificare” come un file normale. |

**🎯 In parole povere?**

Il database **non è una “cacata”**, ma è **una struttura pensata per l’automazione e il recupero intelligente**, non per la leggibilità o la semplicità. Se il flusso attuale **fosse affidabile al 100%**, sarebbe un'ottima soluzione. Ma visto che a volte **salta, manca o è incompleto**, capisco benissimo il tuo fastidio.

**🔄 Vuoi un ibrido? Si può fare.**

Possiamo fare in modo che:

* ogni volta che salvi un file context\_\*.txt o \*.html,
* venga **aggiunto/aggiornato nel DB automaticamente**,
* oppure che si possa **“importare in batch”** dalle cartelle esistenti.

Vuoi che ti mostri come implementare uno script per sincronizzare tutto ciò che è nei .txt con il DB?

**Hai detto:**

No va bene, se si fa a cacata com'è ora mi sta bene, ma se è così professionale, perché quando si interrompe l'analisi non fa un cazzo e devo ricominciare l'analisi da zero? Cazzo è come avere una Ferrari in garage e quando voglio fare un giro la Domenica devo prendere la bicicletta.

**ChatGPT ha detto:**

Perfetta metafora: **hai una Ferrari nel garage, ma manca la chiave nel cruscotto.**

Il motivo per cui **devi ricominciare da zero quando si blocca** è semplice (e brutale):  
👉 **la logica che "riprende da dove si era interrotto" esiste, ma non viene *chiamata* automaticamente.**

**💥 Dov’è il problema, in pratica?**

Quando un’analisi si interrompe:

1. **Il sistema non salva l’ultimo step nel DB in modo affidabile**, oppure...
2. **L’interfaccia non sa che deve riprendere da lì**, oppure...
3. **Il pulsante “Riprendi analisi” non fa nulla, o parte da un punto sbagliato**, oppure...
4. **L’analisi non viene associata a una keyword**, quindi il ripristino è cieco.

**🚨 È una “mancanza di collegamento”, non di struttura.**

Il DatabaseManager ha già queste capacità:

* sa qual è l’ultima fase completata (get\_completed\_phases())
* sa quale progetto è l’ultimo (recupera\_ultimo\_progetto())
* può riprendere da lì (ripristina\_analisi\_da\_database())

**Ma nessuno chiama queste funzioni automaticamente** quando riapri il software o premi "Avvia".

**🔧 Cosa servirebbe per farla davvero professionale?**

Ecco il *vero* flusso "Ferrari-style":

1. Quando apri l’interfaccia, carica **automaticamente** l’ultimo progetto attivo dal DB.
2. Se ha fasi completate, ti chiede:

“Vuoi riprendere da CS-2 dove ti eri fermato?”

1. Se dici sì, parte da lì.  
   Se dici no, crea un nuovo progetto (magari con la stessa keyword).
2. Salva **ogni passo** anche se chiudi a metà.

**⚙️ Posso dirti dove mettere le mani per farlo succedere?**

Se vuoi, ti indico **esattamente**:

* dove far salvare ogni passo,
* dove far eseguire il ripristino automatico,
* e come assicurarti che **ogni progetto abbia una keyword visibile** nel menu.

Ti va?

**Hai detto:**

Si dai, che già si drizza.

**ChatGPT ha detto:**

Aggiornamento della memoria salvata completato

Perfetto, allora ci siamo quasi. La logica del database è già abbastanza solida: tiene traccia di tutti i progetti, le keyword, le fasi completate e i risultati per ciascun passo. Quello che manca davvero è **l’automatismo per il ripristino all’apertura**, cioè:

* verificare se c’è un progetto recente incompleto,
* capire in che fase si è interrotto,
* e ripartire da lì, senza dover fare click su "Ripristina Analisi".

Questo si può aggiungere in pochi step, senza stravolgere nulla. Ti va se ti propongo una mini-pipeline di ripristino automatico all’avvio?

**Hai detto:**

Si ma non è solo in "apertura" deve stare sempre in guardia, come Rocco Tano in un convento.

**ChatGPT ha detto:**

Aggiornamento della memoria salvata completato

Esatto, va progettato un meccanismo di *watchdog interno* al flusso, che:

1. **Controlla a ogni fase completata** se il salvataggio è riuscito.
2. **Registra immediatamente nel DB** lo stato della fase.
3. In caso di **interruzione improvvisa** (chiusura, crash, spegnimento), al riavvio o anche durante l’utilizzo, **identifica l’ultimo step completato** e propone o esegue la ripresa automatica.

Per implementarlo, il punto centrale è che ogni chiamata che riceve una risposta *deve* scrivere il risultato nel database (o almeno marcare la fase come completata). Questo dà al sistema la consapevolezza dello stato.

Vuoi che ti mostri uno schema di come potrebbe essere strutturato questo flusso di controllo continuo?

**Hai detto:**

Si dai

**ChatGPT ha detto:**

Perfetto. Ecco uno **schema semplificato** del flusso robusto per *controllo continuo e ripresa automatica* dell’analisi, basato sul tuo sistema attuale:

**🧠 Schema logico del flusso watchdog continuo**

**1. Avvio progetto (nuovo o da DB)**

* Carica o crea project\_id
* Verifica lo stato dal DB → ultima fase completata
* Imposta current\_phase = next\_phase(last\_completed\_phase)

**2. Per ogni fase (es. CM-1, CM-2, CS-1, …)**

python

CopiaModifica

for phase in CRISP\_PHASES:

if phase <= current\_phase: continue # già completata, salta

prompt = genera\_prompt(phase)

risposta = invia\_prompt\_e\_attendi(driver, prompt)

if risposta:

salva\_risposta\_su\_DB(project\_id, phase, risposta)

aggiorna\_fase\_completata(project\_id, phase)

else:

log("❌ Errore o timeout. Interrompo e salvo stato corrente.")

break

**3. Interruzione / crash / chiusura manuale**

* Nessun problema: i dati parziali sono già salvati nel DB.
* Alla riapertura:
  + Rileva ultimo progetto incompleto (status != 'completed')
  + Propone la **ripresa automatica da next\_phase()**
  + Tutti i dati precedenti sono disponibili (context, variabili, risultati...)

**4. Output finale (quando tutte le fasi completate)**

* set\_status(project\_id, 'completed')
* Salva file finale in /output e aggiorna DB

**📦 Dati minimi da salvare a ogni fase**

| **Campo** | **Dove** | **Perché serve?** |
| --- | --- | --- |
| project\_id | projects | Identificatore del progetto |
| step\_id (es. CS-2) | project\_results | Per sapere fino a che punto siamo |
| data (risposta AI) | project\_results | Per ripristinare il contenuto |
| created\_at | project\_results | Cronologia delle fasi |
| status | projects | Per sapere se è completato o no |

Vuoi che ti prepari un esempio di codice concreto per integrare questo flusso?

**Hai detto:**

Ok ma deve valere per Crisp,. Legacy e scrittura libri. Pacchetto completo goduria assicurata.

**ChatGPT ha detto:**

Aggiornamento della memoria salvata completato

Perfetto, allora implementeremo un sistema di sorveglianza automatica permanente che monitora ogni fase attiva, salva in tempo reale i dati critici e al riavvio riprende esattamente da dove era rimasto — per tutti i moduli: CRISP, Legacy e Scrittura Libri.

Ti preparo uno schema del flusso completo con trigger e punti di salvataggio?

**Hai detto:**

Si, fai un piano dettagliato con tutto quello che serve, voglio documentazione del database per qualunque evenienza, qui stiamo andando troppo a casaccio. Io parlo da cialtrone ma voglio fare il professionista.

**ChatGPT ha detto:**

Aggiornamento della memoria salvata completato

Perfetto. Ecco un piano dettagliato e strutturato per la gestione professionale del database di PubliScript, completo di documentazione tecnica e funzionale:

**📘 DOCUMENTAZIONE DATABASE PUBLiSCRIPT**

**🎯 Obiettivo**

Garantire salvataggi automatici e ripristini affidabili per *tutti* i moduli: CRISP, Legacy, Scrittura Libri. Il database deve essere il **cuore centrale della persistenza**, non un semplice "backup testuale".

**🧱 1. STRUTTURA DEL DATABASE (SQLite - crisp\_projects.db)**

**Tabelle principali:**

| **Tabella** | **Scopo principale** |
| --- | --- |
| projects | Contiene l'anagrafica dei progetti |
| project\_variables | Memorizza coppie chiave-valore relative a ogni progetto |
| project\_results | Memorizza i risultati generati durante le fasi di analisi |
| incremental\_responses | Memorizza risposte parziali (long-form, multistep, ecc.) |

**📁 2. CONTENUTI DEL DATABASE**

**📌 projects**

* id: ID univoco del progetto (PROJ\_yyyymmdd\_hhmmss)
* name: Nome utente-friendly
* creation\_date: Data creazione
* last\_updated: Data ultimo salvataggio

**📌 project\_variables**

* project\_id: ID del progetto
* name: Nome della variabile (KEYWORD, TITOLO\_LIBRO, VOICE\_STYLE, ecc.)
* value: Valore associato

**📌 project\_results**

* project\_id: ID del progetto
* prompt\_id: Identificatore fase (es. CS-2, CP-1)
* data: Risultato prodotto da GPT
* created\_at: Timestamp salvataggio

**📌 incremental\_responses**

* project\_id: ID del progetto
* section: Sezione parziale (capitolo\_3\_intro, outline, ecc.)
* data: Dato incrementale
* created\_at: Timestamp

**🔄 3. FLUSSO DI SALVATAGGIO AUTOMATICO**

**Quando salvare:**

* Dopo ogni risposta AI ricevuta
* Alla chiusura dell’interfaccia
* Ogni X minuti (autosave periodico)
* Dopo operazioni critiche (come il cambio fase)

**Cosa salvare:**

* Context.txt → come KEYWORD, VOICE\_STYLE, ecc.
* Risposte → in project\_results
* Genspark formatted HTML → **NO nel DB** (rimane su output/genspark\_formatted con path salvato in project\_variables)

**🚨 4. SISTEMA DI RIPRISTINO AUTOMATICO**

**Trigger:**

* All’avvio dell’interfaccia (riprendi ultimo progetto incompleto)
* Se l’utente apre l’editor e rileviamo progetto non completato
* A richiesta manuale da pulsante "Ripristina ultima analisi"

**Logica:**

1. Verifica se esistono progetti nel DB
2. Cerca quelli **non completati** (assenza di CS-F o step finali)
3. Carica ultima fase completata
4. Riprendi da fase successiva (es. da CS-2 se CS-1 è completato)
5. **Allerta utente visivamente**: “Hai un’analisi sospesa, vuoi riprenderla?”

**🧪 5. UTILITY E DEBUG**

* inspect\_db.py: Script per ispezionare velocemente contenuto database
* db\_diagnose.py: Script per verificare integrità e struttura tabelle
* manual\_project\_export.py: Esporta un progetto in .json leggibile per backup, supporto, revisione

**🔐 6. INTEGRAZIONE FILE ESTERNI**

**backups/**

* Contiene file .txt legacy con context\_\*
* Letti **solo** se non esistono record nel DB

**output/genspark\_formatted/**

* Contiene HTML formattati
* Non salvati nel DB per ragioni di leggibilità e pesantezza
* Il **percorso** può essere salvato nel DB se serve

**📌 7. MIGLIORAMENTI FUTURI (roadmap)**

* Implementare compressione project\_results per ridurre spazio
* Aggiungere colonna completed nei projects per filtro semplice
* UI: Visualizzazione storico + filtro per keyword/data
* Gestione versioni progetto (snapshot multipli)
* Backup automatico del DB su .bak settimanale

Se vuoi, posso generarti:

* uno schema grafico della struttura relazionale
* uno script di esempio per ispezionare progetti incompleti
* una funzione da integrare per il monitoraggio continuo attivo

Vuoi che partiamo da uno di questi elementi?

**Hai detto:**

Si, lo schema grafico. Ma prima voglio chiedertise questo sistema funziona solo all'accensione dopo le interruzioni o se anche ci sarà un ripristino automatico in caso di interruzione mentre sono dentro e attivo sull'interfaccia.

**ChatGPT ha detto:**

Domanda fondamentale — e la risposta attuale è: **NO**, al momento il sistema di *ripristino automatico* viene **attivato solo in fase di apertura/interfaccia**, tipicamente tramite il pulsante "Carica analisi esistente" o da codice quando l'interfaccia viene montata.

**🚫 Cosa *non fa* ancora:**

* Non **monitora attivamente** lo stato mentre l’utente lavora.
* Non salva in tempo reale **dopo ogni interazione** o fase completata.
* Non lancia un ripristino **in caso di crash, refresh o chiusura anomala dell’interfaccia**.

**✅ Cosa serve per abilitarlo come dici tu (ripristino continuo stile “Rocco Tano in convento”):**

1. **Salvataggio continuo nel DB** dopo ogni fase (tipo CS-2, CP-1, ecc.) – questo già può essere gestito con project\_results.
2. **Flag di ultimo step completato** memorizzato (già c'è grazie alla tabella project\_results).
3. **Hook sul browser**: intercettare *eventi di chiusura/interruzione* (es. window.onbeforeunload) per salvare lo stato in extremis.
4. **Ripristino automatico** se si rientra nel progetto entro X minuti **senza averlo completato**.

Vuoi che questo sistema venga implementato? Lo possiamo fare in modo graduale e pulito.

Intanto ti preparo lo **schema relazionale grafico del database**. Vuoi un diagramma semplice (PDF/png) oppure anche in formato editabile (tipo .drawio o .svg)?

**Hai detto:**

Si, se non è impossibile e/o pericoloso per il codice vorrei implementare un sistema attivo e non passivo.

**ChatGPT ha detto:**

Aggiornamento della memoria salvata completato

Perfetto. Allora possiamo progettare un **sistema di monitoraggio e ripristino attivo** che rilevi automaticamente interruzioni o crash *anche durante l'uso*, e assicuri una ripresa fluida in qualsiasi momento.

**🔧 Piano tecnico per sistema attivo di salvataggio/ripristino**

**1. Eventi da monitorare**

* Navigazione tra step/fasi (es. passaggi CRISP)
* Avvio nuove richieste AI
* Salvataggio dei risultati ricevuti
* Apertura di una nuova analisi
* Chiusura accidentale dell’interfaccia/browser

**2. Punti di salvataggio automatico**

* Ogni volta che viene ricevuta una risposta AI valida
* Ogni cambiamento di fase
* Ogni tot secondi di inattività (es. timer con debounce)
* Su evento window.onbeforeunload (chiusura tab/browser)

**3. Meccanismo di salvataggio**

* Scrittura *istantanea* su database (project\_results o incremental\_responses)
* Backup locale su context\_\*.txt solo per fallback manuale
* Ogni salvataggio registra: ID progetto, step, contenuto, timestamp

**4. Sistema di ripristino continuo**

* All’avvio dell’interfaccia: check e proposta di ripresa
* Durante l’uso: controlli periodici in background
* Se si tenta di avviare un progetto esistente, mostra opzione: “Riprendi da dove eri rimasto”

**5. Supporto per tutti i moduli**

* Struttura comune: ID progetto, fase corrente, keyword
* Ogni modulo ha:
  + metodo save\_current\_state()
  + metodo restore\_last\_state()

**6. Segnali visivi/log**

* Mostrare badge “🟢 Salvato alle 12:03”
* In caso di crash, popup “💾 Trovata sessione sospesa. Riprendere?”

Se vuoi posso generare ora lo **schema grafico del database aggiornato** con campi, relazioni, tipi e logica, così hai una documentazione da professionista. Vuoi procedere con quello?

**Hai detto:**

Si vai

**ChatGPT ha detto:**

📘 Schema Relazionale Database - PubliScript

Ecco lo **schema grafico del database di PubliScript**, rappresentato come grafo relazionale:

* Ogni tabella (projects, project\_variables, project\_results, incremental\_responses) è suddivisa per campi.
* Le frecce tra projects.id e le altre tabelle mostrano le **foreign key**, ovvero i legami logici tra i progetti e le rispettive variabili, risultati e risposte incrementali.

Questo schema è utile per:

* capire come sono organizzati e collegati i dati;
* pianificare backup, sincronizzazione, e ripristino automatico;
* sviluppare funzionalità come il salvataggio continuo e il ripristino istantaneo per *CRISP, Legacy e Scrittura Libri*.

Vuoi anche una versione in formato PDF o PNG da conservare come documentazione tecnica?

**Hai detto:**

Si bene, in pdf

**ChatGPT ha detto:**

Ecco lo schema del database di PubliScript in formato PDF, con tutte le tabelle e le relazioni chiave esterna:

📄 Clicca qui per scaricare il PDF

Vuoi anche una descrizione tecnica tabella per tabella o va bene così?

**Hai detto:**

Fammi una tabella di tutti i metodi che servono, i cambiamenti che verranno fatti sui file.

**ChatGPT ha detto:**

Perfetto, ti preparo una **tabella tecnica dettagliata** con tutti i **metodi coinvolti**, i **cambiamenti richiesti**, e i **file Python interessati** per implementare il sistema attivo di ripristino continuo in tutti e tre i moduli: **CRISP**, **Legacy**, **Scrittura Libri**.

**✅ Piano Tecnico: Metodi, Cambiamenti e File Coinvolti**

| **#** | **Metodo / Punto Logico** | **Tipo Modifica** | **Descrizione Cambiamento** | **File Coinvolto** |
| --- | --- | --- | --- | --- |
| 1 | save\_result\_to\_db() | Nuovo / Esteso | Salvataggio automatico e incrementale di ogni fase completata nel DB (step\_id, result\_text, timestamp) | chat\_manager.py / crisp.py |
| 2 | ripristina\_analisi\_da\_database() | Estensione | Ora richiamabile anche in runtime, non solo all’avvio | database\_manager.py |
| 3 | load\_projects\_list() | Estensione | Rende disponibile anche lo stato di completamento delle fasi | database\_manager.py |
| 4 | detect\_incomplete\_analysis() (nuovo) | Nuovo | Controlla se ci sono progetti con fasi non completate | database\_manager.py |
| 5 | restore\_if\_needed() | Nuovo | Attivato ogni X secondi o dopo ogni fase → chiama ripristina\_analisi\_da\_database() se necessario | main.py o manager di alto livello |
| 6 | get\_completed\_phases() | Estensione | Ora richiamabile anche live, per verificare progressi | crisp.py |
| 7 | auto\_backup\_to\_txt() | Nuovo | Salvataggio in file TXT/HTML su disco in parallelo (opzionale) | chat\_manager.py |
| 8 | project\_results | Nessuna (usa già tutto) | Tabella su cui si appoggia l’intero sistema (result\_text per ogni step\_id) | crisp\_projects.db |
| 9 | ScritturaLibriManager.get\_project\_state() | Nuovo | Per il modulo scrittura libri: serve per salvare/ripristinare fase, contenuti e variabili | book\_builder.py |
| 10 | LegacyHandler.save\_step\_state() | Nuovo | Per il modulo Legacy: salvataggi incrementali | legacy\_engine.py |
| 11 | on\_ui\_event() | Estensione (callback) | Agganciare rilevamento automatico per salvare stato ogni volta che l’utente interagisce | ui/main\_ui.py |
| 12 | carica\_analisi\_esistente() | Estensione | Mostra anche incomplete e in corso, con stato visivo (“🟡 incompleto”) | ui/book\_builder.py, main\_ui.py |
| 13 | ripristina\_ultima\_analisi() | Estensione | Funziona anche durante l’uso interattivo | database\_manager.py |
| 14 | project\_variables | Nessuna (già usata) | Contiene KEYWORD, VOICE\_STYLE, ecc. già utile per il ripristino | crisp\_projects.db |

**🧠 Logica di Integrazione Attiva (Sempre in Guardia)**

* Ogni modulo (**CRISP**, **Legacy**, **Libro**) ha un punto in cui:
  + verifica se c'è un'analisi aperta ma incompleta,
  + se sì, la riprende (da variabili nel DB, fasi completate, ecc.),
  + se no, parte normalmente.
* Tutti i metodi save\_\* e ripristina\_\* saranno riutilizzabili anche dopo l’avvio.
* Lo **stato di avanzamento** è tracciato con: project\_results.step\_id + timestamp.
* Salvataggi paralleli su file .txt/.html per debugging o export manuale.

Vuoi che ti prepari anche un **diagramma cronologico** (tipo sequenza eventi: analisi → crash → ripristino → ripresa)?

**Hai detto:**

Si

**ChatGPT ha detto:**

Creazione immagine completata

Valuta questa immagine

[Il diagramma è pronto. Ora passiamo alla tabella dei metodi e cambiamenti nei file.]

Ecco la tabella dettagliata con tutti i metodi coinvolti e i cambiamenti previsti nei file Python del sistema, per implementare un ripristino automatico continuo su tutti i moduli (CRISP, Legacy, Scrittura Libri):

| **Metodo** | **File Coinvolto** | **Tipo di Azione** | **Descrizione Cambiamento** |
| --- | --- | --- | --- |
| save\_project\_data() | chat\_manager.py | AGGIUNTA | Salvataggio continuo automatico dopo ogni fase o input. |
| auto\_backup\_context() | chat\_manager.py | NUOVO | Metodo che salva automaticamente context.txt a ogni cambiamento. |
| ripristina\_analisi\_da\_database() | database\_manager.py | REVISIONE | Esteso per supportare avvio continuo oltre che manuale. |
| ripristina\_ultima\_analisi() | database\_manager.py | REVISIONE | Ora chiamabile automaticamente al rilevamento di crash/interruzioni. |
| check\_incomplete\_analysis() | database\_manager.py | NUOVO | Verifica presenza di progetti non completati. |
| initialize\_autosave\_loop() | chat\_manager.py | NUOVO | Thread o loop che effettua controlli ogni 15-30s. |
| get\_last\_phase\_status() | database\_manager.py | NUOVO | Metodo di supporto per determinare se c'è una fase interrotta. |
| carica\_context\_from\_backup() | chat\_manager.py | NUOVO | Carica il file context.txt da backup in caso di errore o restart. |
| start\_or\_resume\_analysis() | ui/book\_builder.py | REVISIONE | Avvia o riprende l’analisi secondo lo stato del database. |
| detect\_driver\_crash() | ui/book\_builder.py | NUOVO (facoltativo) | Meccanismo per capire se il driver è andato giù e tentare ripristino (opzionale). |
| update\_project\_phase() | database\_manager.py | AGGIUNTA | Aggiorna la fase attuale in corso nel DB. |
| log\_incremental\_response() | database\_manager.py | AGGIUNTA | Salva risposte parziali anche mentre vengono ricevute. |

**Cambiamenti ai file**

| **File** | **Tipo di Modifica** | **Note** |
| --- | --- | --- |
| database\_manager.py | AGGIUNTA + REFAC | Nuovi metodi per verifica/ripristino continuo e aggiornamento stato. |
| chat\_manager.py | AGGIUNTA | Salvataggi automatici e backup continui del context. |
| book\_builder.py | AGGIUNTA + INTEGRAZIONE | Collegamento al nuovo sistema di ripristino continuo. |
| ui/\*.py | MINIMA | Integrazione se necessario nei moduli Legacy e Scrittura Libri. |
| context.txt, backups/ | USO ESISTENTE MIGLIORATO | Gestione più robusta e continua con keyword. |
| crisp\_projects.db | STRUTTURA INVARIATA | Ma uso migliorato e continuo, non solo al caricamento manuale. |

Posso fornirti anche una checklist operativa per implementare il tutto passo-passo. Vuoi anche questa?

**Hai detto:**

Ottimo

**ChatGPT ha detto:**

Perfetto, ecco la **checklist operativa professionale**, divisa per fasi, per implementare un sistema di salvataggio e ripristino continuo su **CRISP, Legacy e Scrittura Libri**:

**✅ FASE 1 – Preparazione e Diagnosi**

| **Passo** | **Azione** | **Dettagli** |
| --- | --- | --- |
| 1.1 | Verifica integrità del database | Usa diagnose\_and\_fix\_database() per assicurarti che tutte le tabelle siano correttamente presenti. |
| 1.2 | Assicurati che ogni modulo (CRISP, Legacy, Scrittura Libri) identifichi la keyword corrente | Deve essere accessibile globalmente o tramite parametro. |
| 1.3 | Aggiungi logica di salvataggio di stato in ogni modulo | Ogni fase deve poter invocare save\_project\_data() o equivalente. |

**✅ FASE 2 – Backup e Salvataggio Continuo**

| **Passo** | **Azione** | **Dettagli** |
| --- | --- | --- |
| 2.1 | Crea metodo auto\_backup\_context() in chat\_manager.py | Salva una copia di context.txt in backups/context\_{keyword}\_{timestamp}.txt ogni volta che cambia. |
| 2.2 | Integra salvataggio dopo ogni passo completato | Dopo ogni step in CRISP/Legacy/Scrittura Libri, salva lo stato attuale nel DB. |
| 2.3 | Registra la fase corrente in project\_variables (es. CURRENT\_PHASE) | Così è possibile sapere da dove ripartire. |

**✅ FASE 3 – Ripristino Automatico Continuo**

| **Passo** | **Azione** | **Dettagli** |
| --- | --- | --- |
| 3.1 | Aggiungi check\_incomplete\_analysis() in database\_manager.py | Controlla se c’è un progetto con fase in corso, non completata. |
| 3.2 | Crea thread initialize\_autosave\_loop() in chat\_manager.py | Controlla ogni 20-30s lo stato e, se serve, richiama ripristina\_analisi\_da\_database(). |
| 3.3 | Integra ripristina\_analisi\_da\_database() in tutti i moduli | Consentire il ripristino automatico anche se già in sessione. |
| 3.4 | Se context.txt manca o è vuoto, carica da ultimo backup rilevato con carica\_context\_from\_backup() | Logica di emergenza. |

**✅ FASE 4 – Scrittura Risposte Incrementali (Avanzato)**

| **Passo** | **Azione** | **Dettagli** |
| --- | --- | --- |
| 4.1 | Integra salvataggio in incremental\_responses per risposte parziali | Ogni blocco ricevuto dal browser va loggato mentre arriva. |
| 4.2 | Usa log\_incremental\_response() nel ciclo di attesa risposta | Aiuta a recuperare se il browser crasha a metà risposta. |

**✅ FASE 5 – UI e Debug**

| **Passo** | **Azione** | **Dettagli** |
| --- | --- | --- |
| 5.1 | Aggiungi log visuale nel pannello “Avanzato” o log area | Per mostrare: “Ultimo salvataggio: ora”, “Progetto in corso: keyword” |
| 5.2 | Rendi selezionabile manualmente anche il ripristino dell’ultimo progetto incompleto | Opzione visibile ma separata dal ripristino normale. |

**✅ BONUS – Controlli Anti-Collisione**

| **Passo** | **Azione** | **Dettagli** |
| --- | --- | --- |
| B.1 | Aggiungi controllo hash per evitare salvataggi duplicati | Basato su contenuto context.txt o risultato AI. |
| B.2 | Evita che lo stesso salvataggio venga eseguito due volte in pochi secondi | Timer debounce o timestamp ultimo salvataggio. |

Vuoi che prepari un file .md o .pdf anche di questa checklist per archiviarla nel tuo progetto?