IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 11, November 2026

CHRONICLE AI: INTELLIGENT DOCUMENT AND KNOWLEDGE MANAGEMENT SYSTEM

Bhavana Patil[1], Manoj. V. Nikum[2], Dr. D.M. Patel[3]

MCA, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[1]

HOD MCA & Assistant Professor, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[2]

Principal, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[3]

Abstract: Digital documents such as research papers, reports, meeting records, and academic notes contain valuable information, but retrieving a specific fact or understanding a lengthy file can require considerable manual effort. This paper presents Chronicle AI, a proposed web-based document and knowledge management system that combines document processing with language-model assistance. The system is designed to accept PDF and plain-text (TXT) files, extract readable content, generate concise summaries, answer questions using relevant passages from a selected document, extract key points into editable notes, identify dated events, and organise these outputs through a personal knowledge dashboard. The proposed implementation uses Python Flask for server-side application logic, HTML, CSS, Bootstrap, and JavaScript for the user interface, Groq API for language-model inference, Supabase for authentication and persistent storage, and PyMuPDF or pypdf for PDF text extraction. A retrieval-augmented generation approach is proposed for document chat so that answers are conditioned on retrieved source passages and can indicate when the available evidence is insufficient. The design also addresses upload validation, user-level access control, secret management, error handling, and limitations involving scanned PDFs, context length, and generated inaccuracies. Since the supplied project report does not contain a running-system test log or measured performance data, this paper describes the architecture, expected behaviour, and validation plan without claiming unverified experimental results.

Keywords: Artificial Intelligence, Document Management, Retrieval-Augmented Generation, Text Summarisation, Question Answering, Knowledge Management, Flask.

I. INTRODUCTION

Digital documents have become a major source of academic and organisational knowledge. Research papers, project reports, specifications, meeting minutes, and lecture notes often contain facts, definitions, decisions, dates, and action items that users need to revisit. When these materials are stored as separate files, locating and understanding relevant information may involve opening multiple documents and reading them manually. Filename-based search is useful for locating files but does not directly answer questions about their contents.

Chronicle AI is proposed as a unified workspace for uploading PDF and TXT documents and performing several knowledge-oriented tasks on the extracted text. The system aims to reduce repetitive reading by providing document summaries, natural-language question answering, key-point extraction, timeline generation, and a personal document dashboard. Unlike an unrestricted chatbot, its document-chat feature is designed to use passages retrieved from the selected document as evidence for its responses.

The main contributions of the proposed system are: (1) integration of multiple document-understanding tasks in one application; (2) a document-grounded question-answering workflow; (3) structured extraction of notes and dated events; and (4) a modular web architecture with user authentication and persistent storage.

II. LITERATURE REVIEW

A. Conventional Document Management

Traditional document management applications focus on file storage, metadata, access control, and retrieval. Keyword search can locate exact terms quickly, but it may fail when a user expresses the same idea using different words. Manual reading provides human judgement and context, although it becomes timeconsuming for large document collections. These limitations motivate an integrated approach that supports both retrieval and interpretation.

B. Language Models and Summarisation

Large language models can generate summaries, transform unstructured text into structured information, and respond to natural-language questions. Their output depends on the source text, model, and instructions. A fluent response is not necessarily supported by a particular document; therefore, generated summaries and extracted information should be checked against the original source. Chronicle AI treats AI-generated notes and timelines as editable assistance rather than unquestionable facts.

C. Retrieval-Augmented Question Answering

Retrieval-augmented generation (RAG) combines a retrieval stage with a text-generation stage. Relevant passages are selected from a source collection and supplied to the language model as context. This pattern can reduce unsupported document-specific answers when retrieval is accurate and the model is instructed to remain within the supplied evidence. The proposed minimum viable product can begin with text chunking and keyword-based ranking; semantic embeddings and vector similarity are considered future improvements.

III. PROPOSED SYSTEM AND METHODOLOGY

A. System Architecture

Chronicle AI follows a layered architecture consisting of a browser-based presentation layer, a Flask application layer, document-processing services, an AI-inference service, and Supabase persistence. The browser communicates with Flask through application routes. Flask validates the session, request, file type, file size, and document ownership before invoking the relevant service. The document service extracts text, the AI service communicates with the Groq API, and the persistence service stores document metadata and generated outputs. Secret API credentials remain on the server and are not embedded in browser-side code.

The proposed technology stack is summarised as follows: Python Flask provides routing and service orchestration; HTML, CSS, Bootstrap, and JavaScript provide the responsive interface; Groq API provides hosted language-model inference; Supabase provides authentication and relational data storage; and PyMuPDF or pypdf extracts text from supported PDF documents. A single primary PDF library should be selected during implementation and tested with representative files.

B. Document Ingestion and Text Extraction

The user first registers or signs in and uploads a PDF or TXT file. The server validates the extension, size, and content rather than trusting the filename alone. For PDF files, readable text is extracted using the selected PDF library. TXT files are decoded and normalised. The application checks whether the extracted text is empty or unusually short and reports an appropriate warning. Scanned PDFs may contain page images rather than selectable text; OCR is not assumed to be available in the initial version.

C. Retrieval and Document Chat

For question answering, the extracted document text is divided into manageable, potentially overlapping chunks. Each chunk retains document identity and, where available, page or section metadata. When a user submits a question, the backend confirms document ownership, ranks the chunks for relevance, and selects the most useful passages. These passages are included in the model prompt with an instruction to answer only from the supplied evidence. If no relevant evidence is found, the system should report that the document does not provide enough information instead of inventing an answer.

D. AI-Assisted Knowledge Features

The summarisation feature requests a concise overview, key points, important findings, and action-oriented information when present in the source. The smart-notes feature extracts concepts, definitions, names, dates, and action items into structured editable entries. The timeline feature identifies dated events and arranges reliably parsed dates in chronological order. Ambiguous or relative dates are flagged for review rather than silently converted. The dashboard provides access to uploaded documents, summaries, notes, timeline events, and previous document conversations.

E. Data Management and Security

The proposed database entities include documents, summaries, notes, conversations, messages, and timeline events. Records should be associated with an authenticated owner and the relevant document or conversation. Route-level authorisation and database Row Level Security (RLS) policies should prevent one user from accessing another user's information. Upload validation, secure session settings, CSRF protection, environment-based secret configuration, safe error messages, and controlled rendering of model output are important security requirements.

IV. IMPLEMENTATION DESIGN

The application is organised into separate modules to keep responsibilities clear. The main Flask application defines routes and session checks. A document service handles upload validation, PDF/TXT extraction, and text normalisation. A Groq service centralises prompts, model configuration, request timeouts, and response validation. A Supabase service manages database operations. HTML templates render the pages, while CSS, Bootstrap, and JavaScript support navigation, chat submission, loading indicators, and other interface interactions.

The proposed user workflow is: (1) user authentication; (2) PDF/TXT upload and validation; (3) text extraction and document registration; (4) selection of a task such as summary, chat, notes, or timeline; (5) AI processing using the document text or retrieved passages; (6) validation and persistence of the output; and (7) review of saved results from the dashboard. The design allows the PDF library, language model, or database service to be changed without rewriting the complete application.

V. RESULTS AND DISCUSSION

The project report defines the intended features and validation procedures but does not provide a running application, execution logs, completed test records, or measured performance values. Accordingly, no accuracy percentage, response-time result, or successful test rate is claimed in this paper. The expected functional behaviour is that supported PDF and TXT files are accepted, unsupported files are rejected with a safe message, readable text is extracted, and generated outputs are linked to the source document and authenticated user.

A practical evaluation should include digitally generated PDFs, multi-page PDFs, TXT files, scanned or empty PDFs, corrupted files, unsupported extensions, and documents containing dates and action items. Functional tests should cover authentication, upload, summarisation, answerable and unanswerable questions, note extraction, timeline sorting, dashboard persistence, and cross-user access denial. AI output quality should be evaluated separately by checking factual support, completeness, relevance, uncertainty handling, date accuracy, and response time. The actual-result fields must be completed only after tests are run on the implemented application.

Important limitations include incomplete extraction from scanned PDFs, model context-window limits, possible hallucinations, API outages or rate limits, and ambiguity in date expressions. These risks can be reduced through text-quality checks, chunking, evidence retrieval, explicit uncertainty instructions, structuredoutput validation, bounded retries, and source excerpts that users can verify. The system should be treated as an assistance tool and not as an authority for high-impact decisions.

VI. CONCLUSION AND FUTURE SCOPE

Chronicle AI proposes an integrated approach to document and personal knowledge management by combining summarisation, document-grounded question answering, key-point extraction, timeline generation, and a personal dashboard. The proposed Flask-based architecture connects document processing, languagemodel inference, and Supabase persistence through modular services. Its central reliability principle is to ground document-specific answers in retrieved passages and to make generated outputs reviewable by users.

Future work may include OCR for scanned documents, support for DOCX, HTML, spreadsheets, and presentations, semantic embeddings and vector search, background task processing, document versioning, tags and folders, calendar export, richer source citations, and automated regression testing. Further improvements may address rate limiting, audit logging, retention controls, accessibility, containerised deployment, and monitoring. These enhancements should be prioritised after the core workflow has been implemented and evaluated using genuine test evidence.

ACKNOWLEDGMENT

The author expresses sincere gratitude to the project guide, faculty members, and the Department of Master of Computer Applications for their guidance and support. The author also acknowledges the developers and maintainers of Python, Flask, Bootstrap, PyMuPDF/pypdf, Supabase, and the Groq API for their documentation and development resources. The institute and guide details should be completed before final submission.

REFERENCES

[1] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems, vol. 33, 2020.
[2] Pallets Projects, "Flask Documentation." Available: https://flask.palletsprojects.com/
[3] Groq, "Groq API Documentation." Available: https://console.groq.com/docs
[4] Supabase, "Supabase Documentation." Available: https://supabase.com/docs
[5] PyMuPDF, "PyMuPDF Documentation." Available: https://pymupdf.readthedocs.io/
[6] pypdf contributors, "pypdf Documentation." Available: https://pypdf.readthedocs.io/
[7] Bootstrap Team, "Bootstrap Documentation." Available: https://getbootstrap.com/docs/
[8] OWASP Foundation, "File Upload Cheat Sheet." Available: https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html
[9] OWASP Foundation, "Secrets Management Cheat Sheet." Available: https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html