System Design Document

"Second Brain" AI Companion

Name: Harshal Devi

Email: deviharshal13@gmail.com

Phone: +1 469-348-5652

1. Introduction

The Second Brain AI Companion is a personal knowledge management system designed to ingest, persist, and retrieve user-owned information across multiple modalities, including documents, audio recordings, images, web content, and free-form text notes. The system serves as a long-term, searchable memory layer that allows users to query their personal knowledge using natural language and receive precise, citation-backed answers.

The primary objective of the system is to deliver **highly grounded question answering**, where every response is synthesized strictly from user-provided data and can be traced back to specific source segments such as document pages, timestamps, or extracted text regions. The architecture emphasizes correctness, traceability, scalability, and privacy, ensuring that the system remains reliable even as individual knowledge bases grow to thousands of documents.

---

2. System Goals and Design Principles

2.1 Design Objectives

The system architecture is guided by the following core objectives:

- **Multi-Modal Ingestion**
  Seamlessly process heterogeneous data sources including PDFs, Markdown files, plain text notes, audio recordings, images, and web pages.
- **Grounded Answer Synthesis**
  Ensure that all generated answers are derived exclusively from stored user data and accompanied by explicit citations.
- **Hybrid Retrieval Accuracy**
  Combine semantic similarity search with keyword-based lexical search to maximize recall while maintaining precision.
- **Temporal Awareness**
  Preserve and utilize time-based metadata to support queries such as “last meeting,” “earlier this month,” or “during Q2.”
- **Scalability and Privacy**
  Support large personal knowledge bases while enforcing strict user-level data isolation.

2.2 Core Design Principles

- **Separation of Concerns**
  Clear boundaries are maintained between ingestion, processing, storage, retrieval, and generation layers.

- **Asynchronous First**
  All compute-intensive or long-running tasks are executed asynchronously to avoid degrading user-facing performance.
- **Traceability by Construction**
  Every derived artifact such as chunks, embeddings, or answers maintains references to its original source.
- **Extensibility**
  The system is structured to allow additional modalities, retrieval strategies, or models to be integrated without architectural rewrites.

---

3. High-Level Architecture

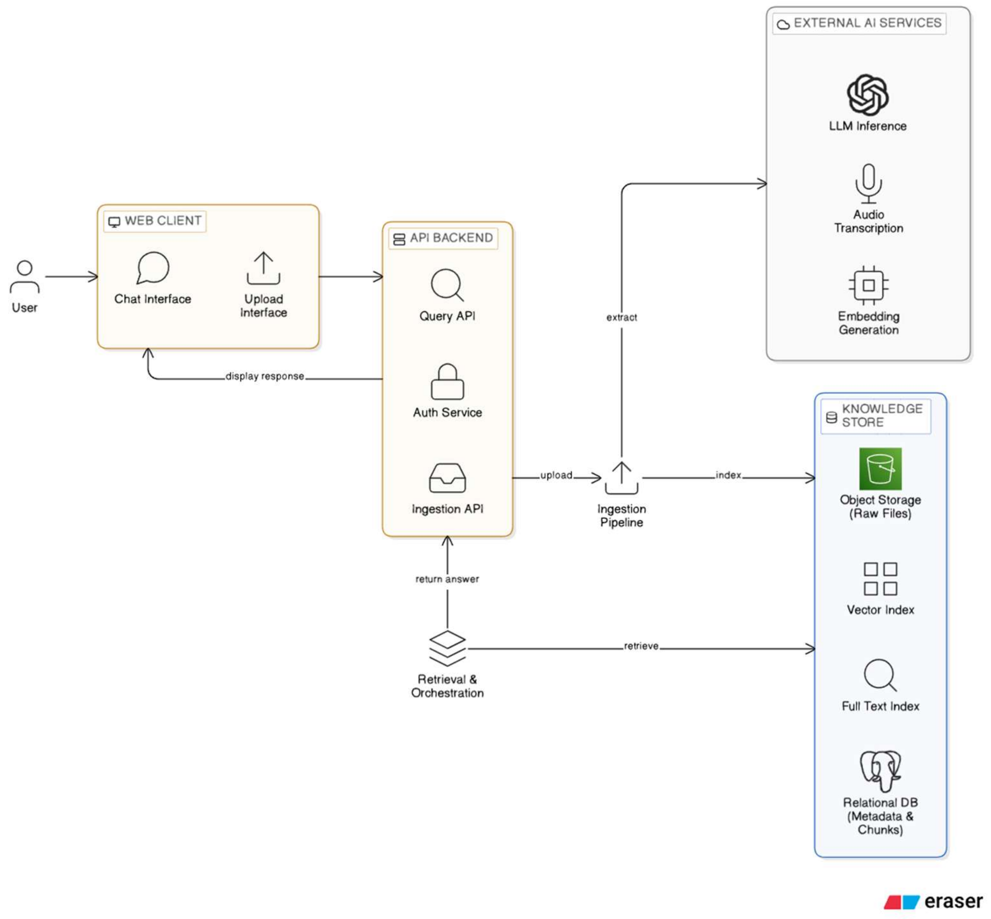

Diagram 1 – High-Level System Architecture

3.1 Architectural Overview

At a high level, the system consists of a Web Client, a centralized API Backend, an Asynchronous Ingestion Pipeline, a persistent Knowledge Store, and a Retrieval and Orchestration layer. External AI services are used selectively for speech transcription, embedding generation, and large language model inference.

All user interactions are mediated by the API Backend, which acts as the system's control plane for authentication, authorization, and request routing.

3.2 Client Interface

The Web Client provides two primary capabilities:

- A file and content upload interface supporting multiple modalities
- A chat-based conversational interface for querying stored knowledge

The client is intentionally thin and does not perform any data processing, retrieval, or inference locally.

3.3 API Backend

Responsibilities

- Authenticate users and enforce authorization
- Accept ingestion requests and enqueue background jobs
- Route query requests to the retrieval layer
- Return finalized responses to the client

The backend is stateless, enabling horizontal scaling under increased load.

4. Multi-Modal Data Ingestion Pipeline

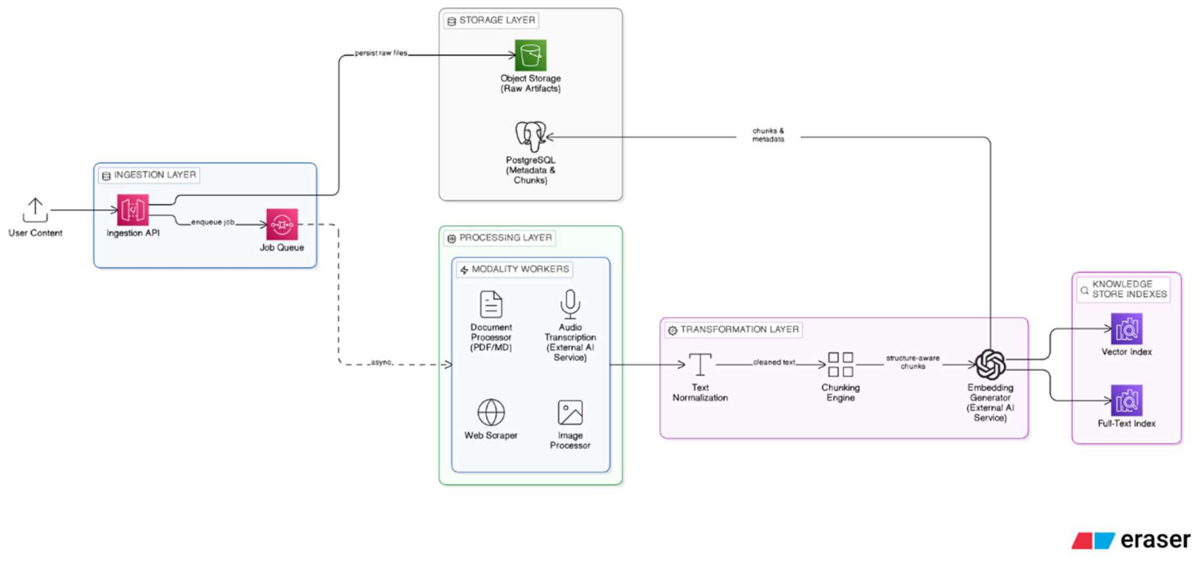

Diagram 2 – Multi-Modal Data Ingestion Pipeline

4.1 Ingestion Overview

The ingestion pipeline transforms raw user content into structured, searchable representations. To ensure responsiveness and fault tolerance, ingestion is fully asynchronous and decoupled from user-facing workflows.

Each upload is tracked as an independent ingestion job with explicit lifecycle states.

4.2 Common Ingestion Stages

Regardless of modality, all content follows a standardized processing sequence:

1. **Receive and Persist**
   Raw artifacts are immediately persisted to object storage. A metadata record is created and an ingestion job is queued.
2. **Extraction and Normalization**
   Modality-specific processors extract text and metadata. Boilerplate is removed, encodings are normalized, and canonical titles are generated.
3. **Chunking and Embedding**
   Extracted text is split into retrieval-friendly chunks with stable boundaries. Each chunk is converted into a vector embedding.
4. **Indexing and Validation**
   Chunks, embeddings, and metadata are indexed into the Knowledge Store. Validation checks are performed before marking the job complete.

4.3 Modality-Specific Processing

Audio Content

Audio files are transcribed using an external speech-to-text service. Transcripts include segment-level timestamps to enable time-scoped queries. Chunks are aligned to speech segments rather than arbitrary text lengths.

Documents

PDF and Markdown files are parsed using structure-aware extractors. Chunking respects headings, pages, and paragraphs with controlled overlap to preserve context.

Web Content

HTML content is fetched with timeout and policy enforcement. A readability extractor isolates primary content. Fetch timestamps are recorded to preserve historical context.

Images and Text Notes

Text notes use paragraph-based chunking. Images are enriched using OCR and caption generation, enabling semantic retrieval over visual content.

4.4 Orchestration and Reliability

- **Idempotency**
  Stable content hashes and chunk identifiers prevent duplication during retries.
- **Retries and Dead-Letter Handling**
  Transient failures are retried automatically. Persistent failures are logged and surfaced for inspection.
- **Progress Tracking**
  Each ingestion job maintains granular status markers, enabling real-time user feedback.

---

5. Knowledge Store Design

5.1 Storage Layers

The Knowledge Store is composed of four coordinated layers:

- **Object Storage**
  Stores raw uploaded artifacts.
- **Relational Database**
  Stores metadata, chunk text, ingestion status, and access control fields.
- **Vector Index**
  Stores embeddings for semantic similarity search.
- **Full-Text Index**
  Stores tokenized text for keyword-based retrieval.

5.2 Data Model

- **Documents**
  Track source type, origin, creation time, ingestion time, and processing state.
- **Chunks**
  Represent atomic retrievable units. Store text, structural offsets, temporal ranges, and ownership identifiers.
- **Embeddings**
  Map chunk identifiers to vector representations along with model version metadata.

This layered model ensures that every embedding and answer can be traced back to a specific raw source.

5.3 Core Schema Definitions

To ensure scalability, traceability, and efficient querying, the system maintains explicit schema definitions for all persisted entities.

- **users**
  - **id (UUID)**
  - **email**
  - **created_at**
- **documents**
  - **id**
  - **user_id**
  - **source_type (audio, pdf, image, web, text)**
  - **source_uri / original_filename**
  - **created_at (event time)**
  - **ingested_at (processing time)**
  - **content_hash (for idempotency)**
  - **status (pending, processed, failed)**
  - **metadata (JSONB)**
- **chunks**
  - **id**
  - **document_id**
  - **user_id**
  - **chunk_index**
  - **text**
  - **token_count**
  - **page_start / page_end**
  - **time_start / time_end**
- **chunk_embeddings**
  - **chunk_id**
  - **embedding_vector**
  - **embedding_model_version**
- **ingestion_jobs**

    - **id**
    - **document_id**
    - **current_stage**
    - **status**
    - **attempt_count**
    - **last_error**
    - **created_at / updated_at**

This schema ensures that every retrieved embedding or generated answer can be deterministically traced back to its originating source.

5.4 Indexing Strategy

The system employs a hybrid indexing approach:

- **Vector Index**
    - Approximate nearest neighbor search (HNSW or IVF).
    - Used for semantic similarity retrieval.
- **Full-Text Index**
    - Tokenized text indexed using inverted indices.
    - Used for exact keyword and identifier matching.

This dual-index design enables high recall while preserving precision.

---

6. Temporal Data Modeling and Query Handling

6.1 Event Time vs Ingestion Time

The system distinguishes between two types of timestamps:

- Event Time: When the content was originally created or occurred.
- Ingestion Time: When the system processed the content.

This separation ensures accurate historical querying even when content is uploaded retroactively.

6.2 Timestamp Assignment by Modality

- **Audio Content**
    - Chunk timestamps are derived from transcript segment offsets.

    - If recording start time is known, absolute timestamps are computed.
- **Documents**
    - File metadata timestamps are preferred.
    - Upload time is used as a fallback.
- **Web Content**
    - Fetch time is always recorded.
    - Published time is parsed when available.
- **Images**
    - EXIF DateTimeOriginal is prioritized.
    - Upload time used when EXIF data is missing.

6.3 Query-Time Temporal Resolution

Natural language time expressions such as “last week” or “in March” are normalized into structured time ranges. During retrieval:

- Chunks whose time ranges overlap the query window are included
- Ranking incorporates temporal proximity as a scoring signal
- Final responses surface explicit dates, pages, or timestamps in citations

---

7. Retrieval and Query Processing

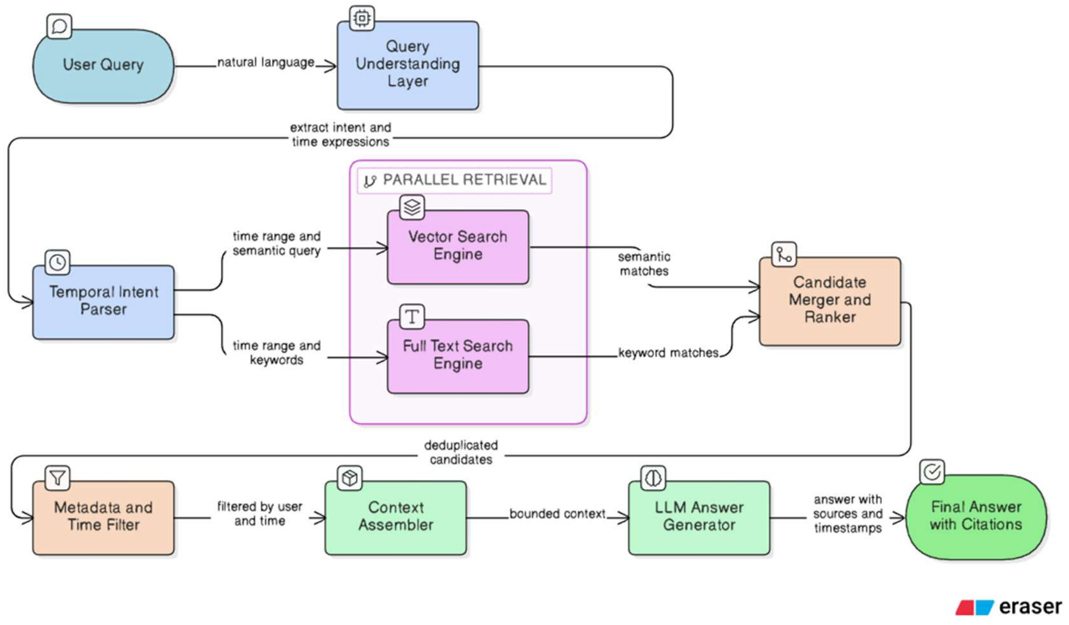

Diagram 3 – Retrieval and Query Processing Pipeline

7.1 Query Understanding

Incoming queries are analyzed to extract intent, semantic meaning, and temporal expressions. Time-based phrases are normalized into structured time ranges.

7.2 Hybrid Retrieval Strategy

The system executes two parallel retrieval paths:

- **Semantic Retrieval**
  Uses vector similarity search to identify conceptually relevant chunks.
- **Lexical Retrieval**
  Uses full-text search to identify exact keyword or identifier matches.

Results are merged and deduplicated at the chunk level.

7.3 Filtering and Reranking

- User-level ownership filters enforce privacy.
- Temporal constraints are applied.
- Candidates are ranked using relevance and recency signals.
- A reranking stage selects the highest-quality chunks within the LLM context budget.

7.4 Answer Generation

The curated context is passed to a large language model with strict grounding instructions. The model is required to:

- Answer only using provided context
- Include structured citations
- Explicitly state when information is unavailable

The final response is returned to the API Backend and then to the client.

---

8. Caching and Performance Optimization

Caching is applied selectively to improve performance without compromising correctness or privacy.

- **Query Result Cache**
  Stores finalized answers scoped by user, normalized query, and time range.
- **Embedding Cache**
  Avoids recomputing embeddings for identical content or repeated queries.

- **Chunk and Metadata Cache**
  Reduces repeated database access for frequently retrieved chunks.

Cache invalidation is triggered by ingestion completion and deletion events.

---

9. External AI Services

External AI services are used for:

- Speech transcription
- Embedding generation
- Large language model inference

These services are abstracted behind clear interfaces, allowing future replacement or self-hosting if required.

---

10. Security and Privacy

- All requests are authenticated and authorized at the API level.
- User data is strictly isolated across all storage and retrieval paths.
- Data is encrypted in transit and at rest.
- Deletion cascades remove all derived artifacts.

10.1 Cloud-Hosted vs Local-First Deployment

A cloud-hosted deployment enables scalable ingestion, low-latency retrieval, and centralized updates. However, it requires strong access control and encryption guarantees.

A local-first alternative provides maximum data sovereignty and offline access but is constrained by local hardware limits and introduces synchronization complexity. The system architecture remains compatible with hybrid approaches, such as local storage of raw files with optional cloud-based embeddings and inference.

---

11. Scalability Considerations

- Asynchronous ingestion supports horizontal scaling.
- Stateless backend services enable replication.
- Approximate nearest neighbor search ensures low-latency retrieval.
- External services isolate compute-heavy workloads.

---

12. Design Tradeoffs

Asynchronous Ingestion vs Synchronous Processing

- Asynchronous ingestion was chosen to ensure a responsive user experience and allow the system to scale under heavy ingestion workloads. While synchronous processing would make newly uploaded content immediately searchable, it would significantly increase request latency and reduce overall throughput.

Hybrid Retrieval vs Single Retrieval Strategy

- A hybrid approach combining semantic and keyword retrieval was selected over a single retrieval method to improve accuracy across diverse query types. Although this increases system complexity, it prevents failures on exact identifiers and paraphrased queries.

Chunk-Level Indexing vs Document-Level Retrieval

- Indexing content at the chunk level enables precise citations, fine-grained relevance scoring, and accurate temporal alignment. This comes at the cost of increased metadata volume and indexing overhead, which was deemed acceptable given the system's emphasis on grounded answers.

External AI Services vs Self-Hosted Models

- External AI services were chosen to reduce operational complexity and accelerate development. While this introduces third-party dependencies, the architecture remains modular and allows for future self-hosted deployments if required.

---

13. End-to-End Workflow Summary

1. User uploads content via the client.
2. API Backend enqueues ingestion jobs.
3. Content is processed, chunked, embedded, and indexed.
4. User submits a query.
5. Hybrid retrieval selects relevant context.
6. LLM generates a grounded, cited answer.
7. Response is returned to the user.