Memoir.ai — Security & Compliance Handbook

Platform Security, Privacy, and Regulatory Alignment

Document Version: 1.0

Status: Operational Reference

Owner: Security & Compliance

Last Updated: YYYY-MM-DD

----

1. Purpose

This handbook defines the security principles, operational safeguards, and regulatory alignment practices governing Memoir.ai. It serves as the primary reference for engineers, operators, auditors, and compliance reviewers responsible for ensuring user data remains private, protected, and under user control.

----

2. Scope

This handbook covers:

• Data protection mechanisms

• Access control and authentication safeguards

• Encryption standards

• Privacy enforcement

• Compliance alignment

• Audit logging

• Incident response

• Secure data lifecycle management

• Billing and entitlement data safeguards

This handbook applies to all components interacting with Memoir.ai vault or metadata systems.

----

3. Security Philosophy

Memoir.ai is built on the following principles:

• Local-first data ownership

• Encryption by default

• Zero data exfiltration without consent

• User-controlled data lifecycle

• Minimal data exposure

• Transparent and auditable operations

The platform must never assume ownership of user content.

----

4. Threat Model Overview

Primary threat vectors include:

• Unauthorized local access

• Theft of physical storage

• Malicious import files

• Malware memory scraping

• Credential compromise

Mitigation strategies include encryption at rest, vault locking, sandbox parsing, and memory sanitization.

Threats outside application control, such as full kernel compromise, remain out of scope.

----

5. Access Control Standards

Physical Access:

• Vault access restricted by operating system permissions.

• Direct file access must require authenticated OS user.

Logical Access:

• Vault unlock requires strong passphrase.

• Auto-lock after inactivity.

• Renderer processes isolated from file system access.

Remote Access:

• No built-in remote access features.

• Support assistance must occur through user-controlled screen sharing only.

----

6. Encryption Strategy

Database Encryption:

• SQLCipher encryption using AES-256.

• Strong key derivation via PBKDF2 or Argon2id.

Key Handling:

• Passphrases never stored.

• Keys generated at runtime only.

• Sensitive memory buffers cleared immediately after use.

Media Encryption:

• Attachments encrypted individually.

• Decryption keys stored only within encrypted vault.

In-Memory Protection:

• Key material marked non-swappable where possible.

• Buffers sanitized after operations.

----

7. Privacy Enforcement

Telemetry:

• Disabled by default.

• Opt-in diagnostics limited to anonymized performance data.

• Personal content never transmitted.

Data Transmission:

• Narrative generation occurs locally.

• Cloud sync transfers metadata only when enabled.

AI Safety:

• AI must avoid diagnosing or speculating about individuals.

• Sensitive personal data must be redacted in public summaries.

----

8. Data Lifecycle Management

Retention:

• Data stored until user deletes it.

• Optional cleanup policies configurable.

Deletion:

• Event deletion removes database records.

• Media removed when orphaned.

• Vault wipe deletes databases, attachments, and caches.

Export:

• Full export always available.

• Uses open, portable formats.

• Offline export supported.

----

9. Audit Logging

Logged events include:

- Vault unlock attempts
- Data import/export actions
- Configuration changes
- System update events

Audit logs must:

- Exclude personal content
- Avoid participant names or message text
- Use hashes instead of identifiers when needed

Logs automatically rotate after retention thresholds.

----

10. Compliance Alignment

GDPR & CCPA Alignment:

- User acts as data controller.
- Application acts as processing tool.
- Data export supports access rights.
- Vault deletion fulfills erasure requests.
- Portable formats satisfy portability rights.

Zero-Knowledge Compliance:

• Vendor cannot access user data.

• External legal requests cannot expose content stored locally.

----

11. Billing & Metadata Safeguards

Billing data protections include:

• Subscription data separated from vault content.

• Payment operations handled via Stripe.

• Entitlements cached locally for offline operation.

Metadata synchronization excludes private message content.

----

12. Incident Response

Vault Corruption:

1. Attempt repair.
2. Restore backup if needed.

Local Breach Response:

1. Disconnect system.
2. Export critical data.
3. Perform vault wipe.

Security Incident Handling:

• Preserve logs.

• Patch vulnerabilities.

• Issue remediation guidance.

Support teams must never request passphrases.

----

13. Secure Development Practices

Engineering teams must:

• Validate all IPC messages.

• Avoid logging sensitive data.

• Keep dependencies patched.

• Apply schema validation to inputs.

• Enforce least privilege policies.

Security reviews required before major releases.

----

14. Operational Security Monitoring

System monitors:

• Disk exhaustion risks

• Database latency spikes

• Worker memory limits

• Background job failures

Alerts presented locally without leaking private data.

----

15. Backup & Recovery Guidance

Recommended safeguards:

• Automatic backups enabled.

• Users encouraged to export periodically.

• Backups stored on external or encrypted media.

----

16. Security Best Practices Summary

• Use strong vault passphrases.

• Keep operating systems updated.

• Maintain encrypted backups.

• Verify imports from trusted sources.

• Review generated narratives for correctness.

----

17. Conclusion

This handbook establishes operational security and compliance foundations for Memoir.ai. By enforcing encryption, privacy preservation, and strict data ownership, Memoir.ai maintains a secure and compliant environment for personal history preservation.

Memoir.ai — Support Escalation Playbook

Version: 1.0

Status: Product & Market Documentation

Last Updated: YYYY-MM-DD

Severity Definitions

Define incident severity levels.

Escalation Flow

Route incidents from frontline support to engineering.

Diagnostic Data Collection

Collect ingestion logs and environment details.

Resolution Tracking

Maintain resolution timelines and post-incident notes.

Memoir.ai — System Design Whitepaper

Architecture Vision, Design Rationale, and System Strategy

Document Version: 1.0

Status: Technical Whitepaper

Owner: Platform Architecture & Strategy

Last Updated: YYYY-MM-DD

----

1. Executive Summary

Memoir.ai is designed as a privacy-first personal archive platform that consolidates fragmented digital histories into a unified, searchable, and narratively explorable timeline. The system combines local-first architecture, encrypted storage, deterministic ingestion pipelines, and AI-assisted narrative generation while preserving user data sovereignty.

This whitepaper outlines the architectural vision, technical decisions, operational strategies, and long-term system goals enabling Memoir.ai to function as a scalable yet privacy-preserving personal knowledge platform.

----

2. Problem Statement

Modern digital life is fragmented across messaging apps, social networks, emails, media libraries, and devices. Users lack tools to unify, search, and contextualize this history while maintaining ownership and privacy.

Cloud-based aggregation tools often compromise privacy or create proprietary lock-in, preventing users from retaining long-term control over their data.

Memoir.ai addresses this by delivering:

• Unified ingestion across platforms

• Searchable chronological reconstruction

• AI-assisted narrative generation

• Local data ownership

• Vendor-independent export capabilities

----

3. Design Principles

System architecture adheres to the following principles:

Local-first operation

User data ownership

Encrypted persistence by default

Deterministic ingestion pipelines

Traceable AI outputs

Minimal cloud dependency

Long-term portability

Cloud services handle metadata and billing only, never personal content.

----

4. Architectural Overview

Memoir.ai operates as a desktop application composed of:

• Electron main process

• React renderer interface

• Background worker processes

• Encrypted vault database

• Job runner queue system

• AI narrative processing pipeline

• Hybrid search subsystem

• Optional cloud metadata synchronization

The architecture isolates compute-heavy tasks from user interaction to preserve responsiveness.

----

5. Execution Model

The system uses a multi-process model:

Main Process

Manages application lifecycle, IPC routing, and filesystem access.

Renderer Process

Hosts the UI and interacts with backend services through secure IPC bridges.

Worker Processes

Handle ingestion, indexing, AI inference, and export packaging without blocking UI execution.

This separation maintains stability under heavy workloads.

----

6. Data Flow Strategy

Key data flows include:

Ingestion Flow

Archives are parsed, validated, normalized, and stored in encrypted databases.

Narrative Flow

Selected evidence is transformed into citation-backed narratives via local AI inference.

Sync Flow

Subscription and workspace metadata synchronize with cloud services without transmitting personal content.

Each flow emphasizes traceability and recoverability.

----

7. Storage & Security Model

Storage architecture divides data into:

Vault Partition

Encrypted SQLCipher database and media assets.

Config Partition

Non-sensitive application settings.

Security measures include:

• AES-256 encryption

• Ephemeral in-memory keys

• Process isolation

• Strict IPC validation

• Sanitized logging

Personal content never leaves the vault without explicit export.

----

8. AI Narrative System Design

The AI subsystem converts event clusters into narratives while preserving factual grounding.

Pipeline stages include:

• Evidence sampling

• Context construction

• Local inference execution

• Citation mapping

• Hallucination verification

• Narrative version storage

All outputs remain editable and versioned.

----

9. Search Architecture

Search combines lexical and semantic retrieval:

• SQLite FTS keyword search

• Vector embedding semantic search

Results blend scores for accurate retrieval while maintaining encrypted local storage.

----

10. Background Job Strategy

Heavy tasks execute through job queues featuring:

• Priority scheduling

• Retry mechanisms

• Checkpointing

• Crash recovery

• Memory throttling

Jobs resume safely after interruption.

----

11. Performance Strategy

Performance objectives include:

• Fast vault unlock

• Smooth timeline scrolling

• Sub-second keyword search

• Efficient large archive imports

• Controlled memory usage

Optimization techniques include chunked ingestion, caching, and worker throttling.

----

12. Data Portability & Longevity

The platform guarantees long-term access through:

• Open export formats

• Structured JSON schemas

• Media hash indexing

• Narrative markdown mirroring

• Integrity checksum validation

Users can reconstruct archives without vendor tools.

----

13. Cloud Metadata Layer

Cloud components support:

• Subscription management

• Device metadata

• Job health monitoring

Strict row-level security ensures tenant isolation while excluding personal vault content.

----

14. Reliability & Recovery Strategy

Reliability features include:

• Import resumption

• Database integrity checks

• Backup restoration paths

• Worker crash recovery

• Retry and idempotency safeguards

System failures must never permanently block data access.

----

15. Future Architectural Directions

Planned evolutions include:

• Multi-device vault synchronization

• Collaborative shared vaults

• Improved semantic reasoning

• Enhanced indexing and summarization

• Incremental inference improvements

Future changes must preserve backward compatibility and data sovereignty.

----

16. Conclusion

Memoir.ai combines local-first architecture, encrypted storage, deterministic ingestion, and AI-assisted summarization to create a private, durable personal history platform. The system design prioritizes user control, scalability, and verifiable processing, ensuring long-term reliability and portability for personal archives.

Memoir.ai — Technical Architecture Manual

System Design, Component Boundaries, and Execution Model

Document Version: 1.0

Status: Engineering Reference

Owner: Platform Architecture Team

Last Updated: YYYY-MM-DD

----

1. Purpose

This Technical Architecture Manual defines the structural, logical, and operational architecture of Memoir.ai. It provides engineers and system operators with a comprehensive understanding of system components, data movement, processing boundaries, and execution flows required to build, maintain, and evolve the platform.

----

2. Scope

This manual covers:

• System architecture and component boundaries

• Multi-process execution model

• Data ingestion and processing flows

• AI snapshot generation architecture

• Storage and encryption model

• Background job execution

• Search and indexing architecture

• Performance strategies and budgets

• Export and data portability model

• Cloud metadata synchronization layer

• API and IPC communication model

UI design and end-user workflows are out of scope.

----

3. Architectural Philosophy

Memoir.ai is designed as a local-first system prioritizing:

• User data ownership

• On-device processing

• Encrypted storage by default

• Deterministic processing pipelines

• Verifiable AI outputs

• Minimal cloud dependency

Cloud services manage metadata, billing, and sync state only. Personal content remains local.

----

4. System Overview

Memoir.ai operates as a desktop application using Electron and React, with a local backend running inside the Electron environment. Heavy computation executes in background workers to keep the interface responsive.

Primary components:

• Electron main process

• Renderer UI process

• Background worker processes

• Encrypted vault storage

• Job runner and queue manager

• AI narrative engine

• Search and indexing services

• Optional cloud metadata layer

----

5. Multi-Process Execution Model

Main Process:

• Controls application lifecycle

• Manages IPC communication

• Handles filesystem access

Renderer Process:

• Runs React UI

• Displays timeline and narratives

• Communicates via IPC bridge

Worker Processes:

• Perform imports

• Execute AI generation

• Build indexes

• Process media

Workers run at lower priority to preserve UI responsiveness.

----

6. Storage Architecture

Vault Partition:

• SQLCipher encrypted database

• Media storage directories

• Thumbnails and caches

Config Partition:

• Non-sensitive settings

• UI preferences

• Window configuration

Encryption keys exist only in memory while unlocked.

----

7. Data Ingestion Architecture

Import flow stages:

1. User selects archive.
2. Extraction worker parses raw data.
3. Validator checks schema integrity.
4. Normalization converts records to canonical format.
5. Events stored in encrypted database.
6. Errors logged and surfaced.

Ingestion runs via resumable background jobs.

----

8. Background Job Architecture

Jobs execute through a SQLite-backed queue.

Lifecycle:

1. Enqueue
2. Worker dispatch
3. Active execution
4. Completion or failure

Features:

• Priority scheduling

• Retry and checkpointing

• Crash recovery

• Memory throttling

----

9. AI Snapshot Generation Architecture

Narrative pipeline:

1. Evidence sampling from database.
2. Prompt context construction.
3. Local model inference.
4. Citation mapping.
5. Verification checks.
6. Narrative storage with version metadata.

All outputs must map to source evidence.

----

10. Provenance & Versioning

Each event and narrative stores provenance metadata.

Versioning rules:

• Regeneration creates new versions.

• User edits preserved.

• Published versions remain immutable.

• Version browsing supported.

Citation integrity maintained across versions.

----

11. Search & Indexing Architecture

Hybrid search combines:

• SQLite FTS keyword search

• Vector-based semantic search

Workflow:

1. Events normalized.
2. Indices built in workers.
3. Queries combine lexical and semantic scores.

All indices remain encrypted inside vault.

----

12. Performance Architecture

Performance optimizations include:

• Chunked ingestion streams

• Worker thread throttling

• Database page caching

• Prepared SQL statements reuse

• Thumbnail caching

Targets include:

• Search <250ms

• Snapshot generation <8s

• Smooth timeline scrolling

----

13. Export & Data Portability

Exports include:

• Events

• Narratives

• Participants

• Media

• Provenance metadata

Data packaged in structured JSON and organized ZIP bundles. Media indexed by hashes for reconstruction.

Exports work offline without vendor dependency.

----

14. Cloud Metadata Layer

Cloud services manage:

• Subscription entitlements

• Device/workspace metadata

• Job sync health

No personal content stored remotely.

Row-level security ensures tenant isolation.

----

15. Communication Architecture

Communication channels include:

IPC Bridge:

• Secure renderer-to-backend communication

Local REST APIs:

• Non-sensitive metadata operations

Webhooks:

• Stripe billing updates

• Auth workspace creation events

All sensitive operations require vault unlock state.

----

16. Security Boundaries

Security controls include:

• Process isolation

• Encrypted vault storage

• Input validation layers

• Sanitized IPC messaging

• Memory key clearing

Renderer processes have no direct filesystem access.

----

17. Failure Recovery Architecture

System supports:

• Import resumption

• Job retries

• Database integrity repair

• Backup restoration

Failures generate logs and user recovery guidance.

----

18. Operational Best Practices

Recommended practices:

• Maintain backward compatibility in schemas.

• Monitor worker memory usage.

• Test ingestion on large archives.

• Validate search performance across releases.

• Audit encryption and key handling regularly.

----

19. Conclusion

The Memoir.ai architecture balances performance, privacy, and verifiability while supporting scalable ingestion and AI-assisted narrative generation. This manual provides engineering teams with a structural reference for platform development and operational reliability.

Memoir.ai — Unified Product & Technical Master Specification

Version: 1.0

Status: Product & Market Documentation

Last Updated: YYYY-MM-DD

Product Overview

Defines Memoir.ai core capabilities and goals.

Architecture Overview

Describes system structure and execution flows.

Data Model Summary

Canonical entities supporting ingestion and narratives.

Security Model

Vault encryption and privacy guarantees.

Operational Flows

Ingestion, snapshot generation, and export workflows.

Compliance Alignment

Supports user-controlled data lifecycle.

Memoir.ai — User Guidebook

Your Personal History, Organized and Private

Document Version: 1.0

Last Updated: YYYY-MM-DD

Welcome to Memoir.ai

Memoir.ai helps you organize your scattered digital life into a private, searchable timeline. Messages, media, and archives from multiple sources are unified so you can rediscover memories, explore patterns, and generate narrative summaries of your life — all while keeping your data under your control.

This guide explains how to set up, use, and manage Memoir.ai effectively.

----

1. Installing Memoir.ai

Download the installer for your operating system.

macOS:

• Open the downloaded .dmg file.

• Drag Memoir.ai into Applications.

• Launch from Applications.

Windows:

• Open the downloaded .exe installer.

• Follow installation prompts.

• Launch Memoir.ai from the Start menu.

----

2. Creating Your Vault

Your vault is your private archive.

Steps:

1. Launch Memoir.ai.
2. Select “Create New Vault.”
3. Choose a storage location.
4. Create a strong passphrase.
5. Confirm vault creation.

Important:

There is no password recovery. Losing the passphrase permanently locks the vault.

----

3. Importing Your Data

Memoir.ai imports exports from messaging and archive tools.

General Import Steps:

1. Open the Imports tab.

2. Choose a supported source.

3. Select exported data files.

4. Confirm import.

5. Wait for processing to complete.

Imports may take several minutes depending on archive size.

----

4. Navigating the Timeline

The timeline combines all sources chronologically.

Features:

• Scroll to browse events.

• Jump by date using navigation controls.

• Filter by source, person, or time.

• Open events to view full details and media.

Timeline navigation is optimized for large archives.

----

5. Searching Your History

Memoir.ai supports two types of search:

Basic Search:

Search names, places, and keywords.

Semantic Search:

Search by intent or memory description, such as:

~the night we moved

~when work was stressful

Filters help refine results by date, source, or attachments.

----

6. Generating Snapshots

Snapshots create narrative summaries of events or relationships.

To generate a snapshot:

1. Select a timeline range or conversation.
2. Click “Generate Snapshot.”
3. Choose tone and length.
4. Wait for generation.

Snapshots include citations linking back to original events.

----

7. Editing Snapshots

Users remain in control.

Capabilities:

• Edit AI-generated text.

• Regenerate sections or full snapshots.

• Revert to previous versions.

• Export narratives for reuse.

Manual edits are preserved unless explicitly replaced.

----

8. Settings and Preferences

Settings allow customization of behavior and storage.

Common options include:

• Theme selection

• Storage location management

• Privacy controls

• Import defaults

• Snapshot preferences

• Keyboard shortcuts

Changes are applied immediately unless stated otherwise.

----

9. Privacy and Security

Memoir.ai prioritizes privacy.

Key protections:

• Data stored locally by default.

• Encryption protects vault contents.

• No telemetry unless explicitly enabled.

• Users retain full ownership of data.

Cloud features, when enabled, synchronize metadata only, not personal content.

----

10. Exporting Your Data

Users can export their entire archive at any time.

Steps:

1. Open Settings.
2. Select Export Vault.
3. Choose export destination.
4. Wait for packaging to complete.

Exports include events, narratives, and media in standard formats.

----

11. Deleting Data or Vaults

Users may delete individual records or wipe entire vaults.

Vault wipe permanently removes:

• Database files

• Attachments

• Local caches

This action cannot be undone.

----

12. Troubleshooting Basics

Common issues:

Import takes too long:

Large imports require processing time. Close heavy applications.

Vault fails to open:

Verify passphrase and vault location.

Search feels slow:

Reindexing may be running in background.

AI output seems incorrect:

Use corrections or regenerate the snapshot.

----

13. Getting Help

Support teams never request passphrases or private data.

When requesting support:

• Provide app version

• Describe issue steps

• Include error messages if available

----

14. Best Practices

Recommended usage patterns:

• Regularly export backups.

• Keep passphrase stored safely offline.

• Import archives periodically.

• Review generated snapshots for accuracy.

Conclusion

Memoir.ai helps you rediscover and understand your digital past while maintaining full privacy and control. Use this guide as a reference while building your personal archive.

Memoir.ai — UX Specifications & Screen Specifications

Document Control

Document Title: UX Specifications & Screen Specifications

Product: Memoir.ai

Version: v1.0

Status: Draft

Owner: Product & UX Engineering

Last Updated: YYYY-MM-DD

Purpose

This document provides implementation-level UX specifications for Memoir.ai screens previously lacking detailed definition. It establishes consistent UI behavior, interaction patterns, animation language, and routing structure required for engineering execution.

Scope

This specification applies to Billing, Settings, and Snapshots screens, global layout conventions, routing requirements, accessibility expectations, and error-handling behavior. Visual assets and code implementation are out of scope.

Audience

Frontend engineers, UX engineers, product managers, QA teams, and accessibility reviewers.

Definitions and Terms

Vault: User storage container.

Snapshot: Generated narrative summary of selected events.

Inspector Panel: Optional contextual side panel.

Nebula Motion: Subtle animation system used across the UI.

Timeline Alignment Pulse: Completion confirmation animation.

Overview

Memoir.ai uses a consistent application shell and motion language to ensure predictable navigation and interaction. Each screen supports universal UI states while maintaining calm motion patterns.

Global UX Conventions

Application Layout Zones

• Left Sidebar Navigation

• Top Context Bar

• Main Content Workspace

• Optional Right Inspector Panel

• Modal Layer

• Toast/Notification Layer

Standard UI States

• Loading

• Empty

• Recoverable error

• Fatal error

• Success confirmation

• Background processing indicators

• Offline or vault-locked state

Motion Language ("Nebula")

• Opacity fades: 150–250 ms

• Vertical entry: 4–8 px

• Timeline pulse after completion

• Shimmer loading placeholders

• No rapid motion or flashing

Animations must never delay user action.

Billing Screen Specification

Purpose

Manage subscription plans, payments, invoices, and usage limits.

Core Components

• Plan Summary Card

• Tier Indicator

• Upgrade/Downgrade Controls

• Usage Metrics Panel

• Payment Method Manager

• Invoice History Table

• Cancellation Panel

• Trial Status Indicator

• Renewal Countdown Indicator

Upgrade Flow

1. User selects plan.
2. Comparison modal opens.
3. User confirms.
4. Payment validated.
5. Confirmation displayed.
6. Limits update immediately.

Cancellation Flow

1. User selects cancellation.
2. Confirmation modal appears.
3. User confirms.
4. Plan remains active until billing end.
5. UI reflects pending cancellation.

States

Loading: Skeleton cards.

Empty: No invoices messaging.

Error: Payment failure alert.

Success: Confirmation toast.

Settings Screen Specification

Purpose

Manage configuration, storage, privacy, imports, and preferences.

Tabs

• General

• Vault & Storage

• Imports

• Privacy & Security

• Snapshot Preferences

• Diagnostics

• Keyboard Shortcuts

Tab Responsibilities

General: Theme, landing screen, notifications.

Vault & Storage: Vault path, move vault, storage visualization, export, delete.

Imports: Default folders, deduplication toggle, parser performance.

Privacy & Security: Passphrase, timeout, analytics toggle.

Snapshots Preferences: Tone, summary length, citation automation.

Diagnostics: Logs, index rebuild, vault repair.

Keyboard Shortcuts: Viewer and customization.

States

Loading: Skeleton loaders.

Error: Vault path warning.

Success: Settings saved toast.

Snapshots Screen Specification

Purpose

Generate and manage narrative outputs.

Core Components

• Snapshot List

• Preview Panel

• Event Selector

• Version History Viewer

• Regenerate Button

• Citation Viewer

• Draft Editor

• Export Controls

Snapshot Creation Flow

1. User selects events.

2. Generation begins.

3. Progress indicator shown.

4. Draft appears.

5. User edits or approves.

Regeneration Flow

1. User regenerates snapshot.

2. Previous version archived.

3. New version displayed.

Version Navigation

Users can revert and compare versions with inline diffs.

States

Loading: Placeholder narrative blocks.

Empty: No snapshots message.

Error: Retry option.

Success: Creation confirmation.

Route Structure

Core Routes

/app

/app/timeline

/app/imports

/app/snapshots

/app/settings

/app/billing

/app/jobs

Nested Routes

/app/snapshots/:snapshotId

/app/timeline/event/:eventId

/app/settings/:tab

Parameters

snapshotId, eventId, tab, sourceId

Deep linking must restore scroll and selection state.

Accessibility Requirements

• Full keyboard navigation

• Screen reader labeling

• WCAG AA contrast compliance

• Visible focus states

Error Handling Philosophy

Errors must always provide explanation, recovery action, and safe fallback. Users must never become trapped in blocked states.

Completion Criteria

Specifications are complete when screens implement required states, navigation functions correctly, motion is consistent, and billing/settings flows are validated.

OVERVIEW AND PRODUCT SPECIFICATION

Generated on: 2026-01-30T16:22:34.909Z

FILE: docs/00_META/CHANGELOG.md

Changelog

All notable changes to the Memoir.ai project documentation and specifications.

[1.0.0-Beta.1] - 2026-01-30

Added

- Complete expansion of 60+ core documentation files.
- Established the `ADR` (Architecture Decision Record) framework.
- Defined AI Safety, Evaluation, and Pipeline specs.
- Documented DevOps, Security, and Commercial models.

Changed

- Unified the "Nebula" design system terminology across all UX specs.
- Refined the "Vault" encryption strategy to emphasize SQLCipher usage.

Fixed

- Reconciled missing files identified in the 44-manifest audit.

FILE: docs/00_META/GLOSSARY.md

Glossary — Memoir.ai

This glossary defines technical and business terms frequently used in the Memoir.ai project to ensure consistent terminology across documentation and code.

1. Domain Terms

Vault

The secure, local storage container where all user data and indices are kept. Encrypted via SQLCipher.

Library (Workspace)

A high-level logical grouping of related archives and memories. A user can manage multiple Libraries.

Memory

A normalized, single record extracted from digital exhaust (e.g., a message, a log entry, a post).

Unified Timeline

The chronological feed that stitches together memories from diverse platforms into a single view.

Narrative

An AI-generated or manually written story, summary, or memoir draft derived from raw memories.

Citation

A verifiable link between a Narrative segment and its source Memory in the timeline.

2. Technical Terms

Local-First

Architecture prioritizing local data processing and storage without cloud dependency.

SQLCipher

An extension for SQLite that provides transparent 256-bit AES encryption for database files.

Ingestion Pipeline

The background process of selecting, validating, parsing, and normalizing external data into the Memoir schema.

Job Runner

The background manager (Node.js) responsible for executing long-running tasks like imports and exports.

JSON Schema

Used for the `eventData` field in the `Events` table to support diverse, schema-less metadata for different source types.

FILE: docs/00_META/MANIFEST_TODO.md

Documentation Manifest To-Do List

This list tracks the implementation of documentation files required by the project's `MANIFEST.md` files but not yet present in the directory structure.

[x] 01_PRODUCT

- ☒ ROADMAP/ROADMAP_V1.md
- ☒ ROADMAP/RELEASE_PLAN_V1.md
- ☒ ROADMAP/BACKLOG_EPICS.md
- ☒ REQUIREMENTS/FUNCTIONAL_REQUIREMENTS.md
- ☒ REQUIREMENTS/NON_FUNCTIONAL_REQUIREMENTS.md
- ☒ REQUIREMENTS/ACCESSIBILITY_REQUIREMENTS.md
- ☒ REQUIREMENTS/DATA_INTEGRITY_REQUIREMENTS.md
- ☒ USER_FLOWS/FLOW_0001_CREATE_LIBRARY.md
- ☒ USER_FLOWS/FLOW_0002_IMPORT_SOURCE.md
- ☒ USER_FLOWS/FLOW_0003_TIMELINE_BROWSE_SEARCH.md
- ☒ USER_FLOWS/FLOW_0004_SNAPSHOT_GENERATE_REVIEW.md
- ☒ USER_FLOWS/FLOW_0005_EXPORT_DELETE.md

[x] 02_DESIGN

- ☒ IA/INFORMATION_ARCHITECTURE.md
- ☒ IA/NAV_MODEL.md
- ☒ IA/ROUTES_MAP.md
- ☒ UX_SPECS/SCREEN_SPECS/APP_SHELL.md
- ☒ UX_SPECS/SCREEN_SPECS/AUTH_LOGIN.md
- ☒ UX_SPECS/SCREEN_SPECS/AUTH_SIGNUP.md
- ☒ UX_SPECS/SCREEN_SPECS/BILLING.md
- ☒ UX_SPECS/SCREEN_SPECS/IMPORTS.md
- ☒ UX_SPECS/SCREEN_SPECS/LIBRARY_HOME.md
- ☒ UX_SPECS/SCREEN_SPECS/SETTINGS.md
- ☒ UX_SPECS/SCREEN_SPECS/SNAPSHOTS.md
- ☒ UX_SPECS/SCREEN_SPECS/TIMELINE.md
- ☒ A11Y/CONTRAST_RULES.md
- ☒ A11Y/KEYBOARD_NAV_SPEC.md
- ☒ A11Y/WCAG_CHECKLIST.md

[x] 03_ARCHITECTURE

- ☒ JOBS_WORKERS/JOB_RUNNER_SPEC.md
- ☒ JOBS_WORKERS/QUEUE_MODEL.md

- ☒ JOBS_WORKERS/RETRIES_IDEMPOTENCY.md
- ☒ SEARCH/QUERY_LANGUAGE_SPEC.md
- ☒ PROVENANCE_VERSIONING/CITATION_SYSTEM.md
- ☒ PROVENANCE_VERSIONING/PROVENANCE_MODEL.md
- ☒ PROVENANCE_VERSIONING/SNAPSHOT_VERSIONING_MODEL.md

[x] 04_DATA

- ☒ DATA_VALIDATION/DEDUPE_MERGE_RULES.md
- ☒ DATA_VALIDATION/IMPORT_VALIDATION_RULES.md
- ☒ DATA_VALIDATION/NORMALIZATION_RULES.md
- ☒ EXPORT_FORMATS/EXPORT_SCHEMA.md
- ☒ EXPORT_FORMATS/EXPORT_ZIP_LAYOUT.md

[x] 05_APIS

- ☒ API_OVERVIEW.md
- ☒ AUTHORIZATION_MODEL.md
- ☒ ERROR_TAXONOMY.md

[x] 06_AI

- ☒ SAFETY/SENSITIVE_CONTENT_POLICY.md
- ☒ SAFETY/USER_CONTROL_GUARDRAILS.md
- ☒ SNAPSHOT_PIPELINE/CITATION_INSERTION_RULES.md
- ☒ SNAPSHOT_PIPELINE/SNAPSHOT_GENERATION_SPEC.md
- ☒ SNAPSHOT_PIPELINE/SNAPSHOT_TEMPLATE_LIBRARY.md
- ☒ SNAPSHOT_PIPELINE/VERSIONING_RULES.md
- ☒ EVALUATION/EVAL_PLAN.md
- ☒ EVALUATION/HALLUCINATION_GUARDS.md
- ☒ EVALUATION/QUALITY_RUBRICS.md
- ☒ PROMPTS/SYSTEM/SYSTEM_CITATION_ENGINE.md
- ☒ PROMPTS/SYSTEM/SYSTEM_IMPORT_SUMMARIZER.md
- ☒ PROMPTS/SYSTEM/SYSTEM_SNAPSHOT_WRITER.md
- ☒ PROMPTS/TASK/PROMPT_CHAPTER_DRAFT.md
- ☒ PROMPTS/TASK/PROMPT_RELATIONSHIP_ARC.md
- ☒ PROMPTS/TASK/PROMPT_TIMELINE_SUMMARY.md

[x] 07_SECURITY_PRIVACY

- ☒ ACCESS_CONTROL.md
- ☒ AUDIT_LOGGING.md
- ☒ COMPLIANCE_NOTES.md
- ☒ DATA_RETENTION_DELETION.md
- ☒ ENCRYPTION_STRATEGY.md
- ☒ EXPORT_GUARANTEES.md

- ☒ INCIDENT_RESPONSE.md
- ☒ PRIVACY_MODEL.md
- ☒ THREAT_MODEL.md

[x] 08_BILLING_AUTH

- ☒ AUTH/RBAC_ENTITLEMENTS.md
- ☒ AUTH/SESSION_SECURITY.md
- ☒ AUTH/SUPABASE_AUTH_MODEL.md
- ☒ BILLING/INVOICING_REFUNDS_POLICY.md
- ☒ BILLING/PAYWALL_RULES.md
- ☒ BILLING/STRIPE_PRODUCTS_PRICES.md
- ☒ BILLING/SUBSCRIPTION_STATES.md
- ☒ BILLING/USAGE_METERING.md

[x] 09_DEVOPS_RELEASE

- ☒ DEPLOYMENT_RUNBOOK.md
- ☒ ENVIRONMENTS.md
- ☒ LOGGING_TRACING.md
- ☒ MONITORING_ALERTING.md
- ☒ RELEASE_GATES.md
- ☒ ROLLBACK_PLAN.md
- ☒ SECRETS_MANAGEMENT.md

[x] 10_QA

- ☒ IMPORT_TEST_CASES.md
- ☒ PERFORMANCE_TESTS.md
- ☒ REGRESSION_CHECKLIST.md
- ☒ SECURITY_TESTS.md
- ☒ SNAPSHOT_TEST_CASES.md

FILE: docs/00_META/NAMING_CONVENTIONS.md

Naming Conventions — Memoir.ai

Consistent naming ensures clarity across the multifaceted Memoir.ai codebase and documentation.

1. Documentation

- **Filenames**: Uppercase with underscores (e.g., `PRD_V1.md`, `MODULE_MAP.md`).
- **Folders**: Numeric prefix with uppercase category (e.g., `01_PRODUCT`, `04_DATA`).
- **Headings**: Sentence case for subheadings; Title Case for primary titles.

2. Database (SQLite/SQLCipher)

- **Tables**: Plural, lowercase (e.g., `users`, `memories`, `data_sources`).
- **Columns**: camelCase (e.g., `userId`, `importedAt`, `eventData`).
- **Primary Keys**: Always named `id`.
- **Foreign Keys**: `tableNameId` (e.g., `dataSourceId`).

3. API Endpoints

- **Path structure**: `/api/v1/resource-name`
- **Resource names**: Plural, kebab-case (e.g., `/api/data-sources`).
- **Methods**:
  - `GET`: Retrieval.
  - `POST`: Creation / Job Initiation.
  - `PATCH`: Incremental updates.
  - `DELETE`: Removal.

4. Frontend (React)

- **Components**: PascalCase (e.g., `TimelineItem.jsx`, `JobRunnerDashboard.jsx`).
- **Hooks**: `use` prefix (e.g., `useAuth`, `useTimeline`).
- **Constants**: UPPER_SNAKE_CASE (e.g., `MAX_IMPORT_SIZE`).
- **Variables/Functions**: camelCase (e.g., `handleLogin`, `filteredMemories`).

5. Storage (Local FS)

- **Vault Path**: `~/.memoir/vault.db`
- **Export Paths**: `~/Downloads/Memoir_Export_[DATE].zip`

FILE: docs/00_META/README_DOCS_MAP.md

Memoir.ai Documentation Map

A master index of all architectural, technical, and product documentation.

00_META (Core Standards)

- GLOSSARY.md
- NAMING_CONVENTIONS.md
- CHANGELOG.md

- SCOPE_LOCK_V1.md
- **ADRs**: ADR_INDEX.md

01_PRODUCT (Strategy & Requirements)

- VISION.md
- PRD_V1.md
- USER_PERSONAS.md
- JTBD.md
- SUCCESS_METRICS.md
- **Roadmaps**: ROADMAP_V1.md, RELEASE_PLAN_V1.md
- **Requirements**: FUNCTIONAL_REQUIREMENTS.md, NON_FUNCTIONAL_REQUIREMENTS.md
- **Flows**: FLOW_0001_CREATE_LIBRARY.md, FLOW_0002_IMPORT_SOURCE.md

02_DESIGN (UX/UI System)

- COLOR_SYSTEM.md
- TYPOGRAPHY.md
- A11Y Checklist
- **IA**: NAV_MODEL.md, ROUTES_MAP.md
- **Screens**: TIMELINE.md, SNAPSHOTS.md

03_ARCHITECTURE (Technical Infrastructure)

04_DATA (Persistence Layer)

- CANONICAL_DATA_MODEL.md
- **Supabase**: SCHEMA_OVERVIEW.md, RLS_OVERVIEW.md
- **Exports**: EXPORT_SCHEMA.md

05_APIS (Interface Layer)

- API_OVERVIEW.md
- ERROR_TAXONOMY.md
- **Endpoints**: BILLING_API.md, IMPORTS_API.md

06_AI (Intelligence Layer)

- AI_OVERVIEW.md
- **Safety**: SENSITIVE_CONTENT_POLICY.md
- **Eval**: HALLUCINATION_GUARDS.md

08_BILLING_AUTH (Commercial Layer)

- **Auth**: SUPABASE_AUTH_MODEL.md
- **Billing**: PAYWALL_RULES.md

11_USER_DOCS (Guides)

- GETTING_STARTED.md
- IMPORT_GUIDE.md
- SEARCH_GUIDE.md

12_PROMPT_LIBRARY (AI Configuration)

- **Build Agents**: MASTER_BUILD_PROMPT.md
- **UI Copy**: MICROCOPY_SYSTEM.md

Utility Layers

- **Scripts**: verify_manifests.js, scaffold.js
- **Templates**: TEMPLATE_SPEC.md

FILE: docs/00_META/SCOPE_LOCK_V1.md

Scope Lock (V1 Release)

Finalized feature list for the Memoir.ai V1 Launch.

1. Core Platform

- [x] Local Vault Creation (SQLCipher).
- [x] Multi-source Ingestion (iMessage, WhatsApp, JSON).
- [x] Unified Timeline View (Media + Text).
- [x] Semantic Search (Natural Language Queries).

2. Intelligence Layer

- [x] Archive Snapshots (Yearly/Monthly Summaries).
- [x] Relationship Arcs (Analysis of 1-on-1 contact).
- [x] Citation Hover Previews.
- [x] Hallucination Monitoring.

3. Commercial & Security

- [x] Pro Tier Entitlements (Unlimited Sources).
- [x] Stripe Sync (Supabase Backend).
- [x] AES-256 Encryption-at-rest.
- [x] ZIP Data Export.

4. DEFERRED (V2+)

- [ ] Collaborative Vaults (Shared History).
- [ ] iOS/Android Companion Apps.

- ☐ Social Media Feed Ingestion (Instagram/Twitter).

FILE: docs/00_META/ADR/ADR_0001_STACK_DECISION.md

ADR 0001: Technology Stack Decision

Status

ACCEPTED

Context

Memoir.ai targets high-privacy, local-first digital history. It must handle multi-gigabyte data archives, perform local LLM inference, and provide a premium, cinematic desktop experience while maintaining cross-platform compatibility.

Decision

- **Frontend**: React + Vite (Fast HMR, modern ecosystem).
- **App Shell**: Electron (Native APIs, predictable environment for local LLM).
- **Main Logic**: Node.js/Express.js (Shared IPC layer for complex background jobs).
- **Database**: SQLite with SQLCipher (Local AES-256 encryption-at-rest).
- **AI**: Local LLM (Llama-3 via Metal/CUDA) + VectorDB (ChromaDB or similar).

Rationale

- **Electron** allows bundling local binaries (SQLCipher, LLM engines) which is difficult in a standard browser.
- **SQLCipher** provides the essential "Fortress" security requirement.
- **Node.js** allows us to run heavy ingestion scripts in the background without blocking the UI.

FILE: docs/00_META/ADR/ADR_0002_PROVENANCE_MODEL.md

ADR 0002: Provenance Model

Status

ACCEPTED

Context

Factual accuracy is paramount. Users must trust the AI's "Cinematic Narratives." This requires a way to trace every generated sentence back to its raw origin (message, email, photo).

Decision

Adopt an immutable **Event-Based Provenance Model**:

- Every raw data point is assigned a `UUID` upon ingestion.
- The AI pipeline generates `Citation` records that link `GeneratedFragmentID` to `EventID`.
- Citations are stored as a first-class table, not just text markers.

Rationale

- Decoupling citations from text allows the UI to show rich hover-previews of the source material.
- Immutable IDs ensure that even if a message is deleted, the provenance record can flag the missing evidence rather than silently failing.

FILE: docs/00_META/ADR/ADR_0003_SNAPSHOT_VERSIONING.md

ADR 0003: Snapshot Versioning

Status

ACCEPTED

Context

AI output is stochastic. Users may iterate on prompt parameters or manually edit the narrative. We need to preserve the history of these "Memoirs."

Decision

Implement a **Mutable Head, Immutable History** versioning system:

- Each Snapshot has a `version_tree`.
- Local edits create a "Delta" patch rather than a full copy.
- Automated regenerations create a new branch.

Rationale

- Minimizes disk usage by only storing diffs for large text files.
- Allows "Compare" views where users can see how an AI's interpretation changed across different model versions.

FILE: docs/00_META/ADR/ADR_0004_SEARCH_STRATEGY.md

ADR 0004: Search Strategy

Status

ACCEPTED

Context

The app must search across disparate data types (text, image meta, voice transcripts) with sub-second latency on a local machine.

Decision

Use a **Hybrid Search Engine**:

- **FTS5 (SQLite)**: For high-speed lexical/keyword matching.
- **Vector Search (Local)**: For semantic "Vibe" matching (e.g., searching for "Beach" finds "Ocean").
- **Faceting**: Performed in-memory via the Redux state for the current view.

Rationale

- FTS5 is natively available in SQLite and extremely memory efficient.
- Semantic search provides the "Magic" UX that differentiates Memoir.ai from basic log viewers.

FILE: docs/00_META/ADR/ADR_0005_IMPORT_PIPELINE.md

ADR 0005: Import Pipeline

Status

ACCEPTED

Context

Archived data files (e.g., Apple Privacy Downloads) can be 50GB+. Processing these in a single thread crashes the Electron main process.

Decision

Implement a **Streaming, Multi-Process Worker Pipeline**:

- Main process spawns `IngestionWorker` (Node.js child process).

- Workers stream raw data, parse into `JSONL`, and batch insert into SQLite.
- **WAL Mode**: SQLite must enable Write-Ahead Logging to allow UI reads during heavy ingestion writes.

Rationale

- Streaming prevents "Out of Memory" errors on large ZIP files.
- Child processes protect the UI responsiveness from being degraded by heavy hashing/parsing logic.

FILE: docs/00_META/ADR/ADR_INDEX.md

ADR Index

Summary of all architectural decisions for Memoir.ai.

ID	Title	Status	Date
0001	Stack Decision	ACCEPTED	2026-01-30
0002	Provenance Model	ACCEPTED	2026-01-30
0003	Snapshot Versioning	ACCEPTED	2026-01-30
0004	Search Strategy	ACCEPTED	2026-01-30
0005	Import Pipeline	ACCEPTED	2026-01-30

FILE: docs/01_PRODUCT/JTBD.md

Jobs to be Done (JTBD) — Memoir.ai

Memoir.ai focuses on the core "jobs" users are trying to accomplish when managing their digital lives.

1. Core Jobs

Job 1: Consolidate Digital Fragmentation

- **Situation**: "When I look back at my year, I realize my memories are split between iMessage, Instagram, and two email accounts."
- **Motivation**: "I want to bring all these pieces together in one chronological view."
- **Outcome**: "So that I can see the full context of my life in one place."

Job 2: Protect Personal History (Privacy Lock)

- **Situation**: "When I am handling my most private messages and photos..."

- **Motivation**: "I want a local-first solution that doesn't upload anything to the cloud."
- **Outcome**: "So that I have absolute control and peace of mind over my data."

Job 3: Deriving Meaning (Narrativization)

- **Situation**: "When I have years of raw message data from a significant relationship..."
- **Motivation**: "I want tools to summarize those patterns into a narrative or memoir draft."
- **Outcome**: "So that I can understand how my relationships evolved and reflect on my growth."

2. User Journey Mapping

Phase 1: Onboarding & Setup

1. **Job**: Initialize the Vault.
2. **Action**: Choose storage location and set encryption passphrase.
3. **Outcome**: A secure, encrypted local database is ready.

Phase 2: Multi-Source Ingestion

1. **Job**: Import raw Archives.
2. **Action**: Select iMessage/WhatsApp/Email sources via the Import Wizard.
3. **Outcome**: Files are validated and enqueued for parsing.

Phase 3: Processing & Normalization

1. **Job**: Monitor progress.
2. **Action**: View real-time status in the Job Runner Dashboard.
3. **Outcome**: Data is transformed into a unified format and indexed.

Phase 4: Exploration & Synthesis

1. **Job**: Explore history.
2. **Action**: Browse the Unified Timeline, use Advanced Search, and generate Narratives.
3. **Outcome**: Insights are gained and narrative drafts are created.

FILE: docs/01_PRODUCT/MARKET_RESEARCH.md

Market Research — Memoir.ai

1. Industry Overview

The personal data management industry is shifting toward **Local-First** solutions as consumer awareness of privacy reaches an all-time high.

- **Valuation**: Global personal data management market is valued at ~$5B (2025) with a 12% CAGR.

- **Privacy Sentiment**: 70% of consumers express concerns about data privacy, driving adoption of offline tools.
- **Target Market (TAM)**: Projected to reach $8B by 2030.

2. Competitive Landscape

Memoir.ai competes in the "Digital Storytelling" and "Personal Archive" niche.

Main Competitors

Competitor	Advantages	Disadvantages	Market Position
Journey	Narrative creation, user base.	Cloud-based (Privacy risk).	Leader in Narratives.
Day One	Mobile presence, journaling.	Cloud dependence, limited import.	Leader in Journaling.

SWOT Analysis

- **Strengths**: Local-first privacy, comprehensive narrative generation.
- **Weaknesses**: New entrant, limited initial brand recognition.
- **Opportunities**: Privacy-conscious growth, educational/wellness partnerships.
- **Threats**: Rapid tech changes, established legacy competitors (Day One), increased regulatory scrutiny on data privacy, and potential economic downturns affecting subscription spend.

3. Search Trends (2025-2026)

Search interest in "personal data management" and "local-first applications" is increasing steadily.

Month	Search Volume	Trend	Competition
2025-11	15,000	Increasing	Medium
2025-12	18,000	Increasing	Medium
2026-01	20,000	Increasing	High

4. Monetization & ROI

- **Model**: Subscription with a Freemium tier (Monthly: $9.99 / Annual: $99.99).
- **Projections**: 10,000 users and $1M revenue by Year 3.
- **ROI**: 300% projected ROI with breakeven at 18 months.

FILE: docs/01_PRODUCT/PRD_V1.md

PRD (Product Requirements Document) - V1

Product: Memoir.ai
Status: LOCKED
Version: 1.0.0-V1

1. Executive Summary

Memoir.ai is a privacy-first personal historian that turns fragmented digital archives into cohesive, cinematic narratives. By combining local-first storage with on-device AI, it provides a "Private Time Machine" experience.

2. Core Pillars

- **Radical Privacy**: No user data ever leaves the local machine unless explicitly exported by the user.
- **Narrative Truth**: Every AI claim must be backed by a verifiable citation to raw data.
- **Rich Aesthetics**: The "Nebula" theme provides a premium, immersive discovery experience.

3. High-Level Requirements

Reqs derived from `docs/01_PRODUCT/REQUIREMENTS/FUNCTIONAL_REQUIREMENTS.md`.

4. User Journey

1. **Onboard**: Create vault with strong passphrase.
2. **Ingest**: Import iMessage and Photos archives.
3. **Discover**: Browse the timeline; find forgotten milestones.
4. **Reflect**: Generate a "Year in Review" snapshot.
5. **Preserve**: Export the entire archive for long-term storage.

FILE: docs/01_PRODUCT/SUCCESS_METRICS.md

Success Metrics (KPIs) — Memoir.ai

To track the growth and health of the Memoir.ai ecosystem, we monitor the following Key Performance Indicators (KPIs), as defined in the market specification.

1. User Engagement Metrics

- **Monthly Active Users (MAU)**: Target 5,000 MAU in Year 1.
- **Retention Rate**: Percentage of users who return to the timeline weekly.

- **Import Success Rate**: Ratio of successfully completed import jobs vs. failed ones.
- **Time to Value**: The time from first launch to the first AI-generated summary viewed.

2. Product Quality Metrics

- **Search Latency**: Target < 200ms for 95% of queries on libraries > 10,000 events.
- **Parsing Reliability**: Percentage of detected events successfully normalized.
- **Net Promoter Score (NPS)**: Target > 50, reflecting high user trust in privacy and design.

3. Business & Monetization

- **Annual Recurring Revenue (ARR)**: Target $1M within the first three years.
- **Customer Acquisition Cost (CAC)**: Monitored through organic vs. paid channel performance.
- **LTV / CAC Ratio**: Target > 3.0 for sustainable growth.
- **Conversion Rate**: Percentage of free tier users moving to the premium subscription.

4. Tracking Method

All user-facing metrics are tracked **locally** and can be viewed in the app's internal "Product Health" dashboard. Generic, non-identifiable usage statistics may be shared with users to help them understand their own habits, but no raw data is ever sent to Memoir.ai servers without explicit user consent for specific debugging purposes.

FILE: docs/01_PRODUCT/USER_PERSONAS.md

User Personas — Memoir.ai

1. Primary Audience Overview

The core user base for Memoir.ai consists of tech-savvy individuals who value privacy and are interested in personal reflection and digital archiving.

Attribute	Detail
Age Range	18 - 35 years old
Sector	Personal Data Management, Digital Storytelling
Key Interests	Digital archiving, Personal development, Creative writing, Data privacy, Social media analysis, Relationship management.

2. Core Personas (Archetypes)

The "Digital Archivist"

- **Goal**: Consolidate a lifetime of fragmented digital history into a single, permanent record.
- **Need**: Multi-source integration (iMessage, WhatsApp, Email) and secure, offline storage.
- **Value**: Peace of mind that their digital legacy is protected and organized.

The "Reflective Storyteller"

- **Goal**: Transform raw communication data into meaningful narratives and memoir drafts.
- **Need**: Tools for analyzing communication patterns and generating AI-assisted summaries.
- **Value**: Turning "digital exhaust" into self-understanding.

The "Privacy Advocate"

- **Goal**: Manage personal history without exposing it to cloud providers or telemetry.
- **Need**: Local-first processing and encrypted datastores (SQLCipher).
- **Value**: Complete control and confidentiality of their most sensitive data.

3. User Needs & Pain Points

Needs

- **Secure Storage**: Protection against unauthorized access.
- **Ease of Use**: A guided interface for complex data imports.
- **Relationship Insights**: Deep analytics on communication and emotional trends.
- **Narrative Assistance**: Automated and manual tools for storytelling.
- **Offline Access**: Independence from internet connectivity for data browsing.

Pain Points

- **Fragmentation**: Data scattered across a dozen platforms.
- **Fragmentation Stress**: Emotional distress from an inability to make sense of the past.
- **Privacy Anxiety**: Fear of cloud-based data breaches or surveillance.
- **Complexity**: Raw exports (MBOX, JSON, CSV) are currently unusable for the average user.

FILE: docs/01_PRODUCT/VISION.md

Product Vision

The guiding north star for Memoir.ai.

1. The Problem

Personal history is scattered across dozens of platforms (Apple, Google, Meta). It is fragmented, commercialized, and increasingly inaccessible to the person who lived it.

2. The Vision

"A digital sanctuary for your life's story."

3. Our Values

- **Ownership**: You own the bits. You own the interpretation.
- **Integrity**: We don't guess; we cite.
- **Timelessness**: The software should feel like a library, intended to last decades, not a feed intended to last minutes.

4. Implementation Philosophy

- **Local-First**: The database lives where you live.
- **Design Excellence**: Beauty is a privacy feature; users protect what they love.
- **Zero-Ad Model**: Our only customer is the user.

FILE: docs/01_PRODUCT/ROADMAP/BACKLOG_EPICS.md

Backlog & Epics — Memoir.ai

High-level epics categorized by functional area for V1 and beyond.

Epic: Core Vault & Security

- [ ] Robust SQLCipher passphrase management.
- [ ] Auto-lock timers and session security.
- [ ] Vault integrity repair tools.

Epic: The Ingestion Engine

- [ ] Multi-threaded parser for social media archives (FB/IG).
- [ ] Deduplication logic for overlapping messaging platforms.
- [ ] Media extraction and thumbnail caching optimization.

Epic: Cinematic UI/UX

- [ ] Nebula design system implementation (Violet/Cyan/Magenta).
- [ ] Infinite scroll performance for 50k+ timeline events.
- [ ] Conversation viewer with cross-platform thread reconstruction.

Epic: Local AI & Narratives

- ☐ Citation-grounded Snapshot generation.
- ☐ Versioning and diff-viewer for AI narrative drafts.
- ☐ Relationship graph and emotional trend analysis.

Epic: Commercials & Growth

- ☐ Stripe checkout and subscription state machine.
- ☐ Onboarding wizard with vault creation guide.
- ☐ Local-first success metrics and KPI dashboard.

FILE: docs/01_PRODUCT/ROADMAP/RELEASE_PLAN_V1.md

V1 Release Plan — Memoir.ai

The release strategy for Memoir.ai V1 focuses on stability, privacy verification, and user onboarding.

1. Release Milestones

- **Alpha (Internal)**: Core vault and iMessage import validation.
- **Beta (Closed)**: Selected test group (100 users) focused on high-volume archive stability and AI hallucination checks.
- **RC (Release Candidate)**: Full feature set including billing and help documentation.
- **V1 Global**: Public desktop release on macOS and Windows.

2. Release Gates

A version is released only if:

1. **Encryption**: SQLCipher initialization succeeds on all target platforms.
2. **Performance**: Search latency remains < 200ms on golden test sets (10k+ events).
3. **Privacy**: Audit confirms ZERO external telemetry calls.
4. **AI Safety**: Hallucination guards trigger < 1% of the time on verified snapshots.

3. Distribution Channels

- **Website**: Direct DMG/EXE downloads with auto-updater support.
- **Support**: Embedded user guides and locally-stored troubleshooting logs.

4. Rollback Strategy

In the event of a fatal V1 bug:

- Immediate halt of auto-update distribution.
- Hotfix deployment within 24 hours.
- Instructional guide for users to restore from local encrypted backups.

FILE: docs/01_PRODUCT/ROADMAP/ROADMAP_V1.md

V1 Roadmap — Memoir.ai

This document outlines the strategic phases leading to the V1 launch and beyond, as defined in the market and business specifications.

Phase 1: Foundation (Months 1-2)

- **Core Infrastructure**: Implementation of Electron shell, React boilerplate, and SQLCipher integration.
- **Authentication**: Local-first JWT authentication and secure vault setup.
- **Basic Ingestion**: iMessage and WhatsApp parsers initialized.

Phase 2: Ingestion & Timeline (Months 3-4)

- **Job Runner**: Background processing for large archives.
- **Unified Timeline**: Master chronological feed with infinite scroll.
- **Conversational Logic**: Thread reconstruction and multi-source alignment.

Phase 3: AI & Insights (Months 5-6)

- **Snapshot System**: On-device AI generation with citation grounding.
- **Relationship Analytics**: People-centric insights and communication trends.
- **Advanced Search**: Semantic and keyword-based query engine.

Phase 4: Release & Monetization (Launch)

- **Premium Tier**: Stripe integration and paywall logic.
- **User Guides**: Finalization of onboarding and support documentation.
- **App Packaging**: Code signing and binary distribution for macOS/Windows.

Future Roadmap (Post-V1)

- External API Integrations (Optional Cloud Proxies).
- Mobile Companion App (Read-only view).
- Collaborative Timelines (Shared vaults).

FILE: docs/01_PRODUCT/ROADMAP/V1_ROADMAP.md

V1 Roadmap & Financials — Memoir.ai

1. Development Milestones

Phase	Milestone	Description
Phase 1: Foundation	Secure Auth & Libraries	Supabase Auth, workspace management, and core schema ready.
Phase 2: Ingestion	Ready Pipeline	Single format (CSV/JSON) import pipeline with background jobs functional.
Phase 3: Core UX	Timeline & Search	Chronological view and full-text search deployed.
Phase 4: AI & Launch	Snapshot MVP	AI generation with citations and versioning ready for launch.

2. Financial Projections (First 3 Months)

Based on a target of 1,000 active users and an assumed 10% paid user conversion.

Revenue Projection

- **Paid Subscribers**: 100 users (10% conversion).
- **Revenue**: Based on introductory subscription pricing (to be finalized).

Estimated Costs

- **AI Services**: Per-token costs for Snapshot generation (targeting ≥3 per user).
- **Cloud Infrastructure**: Supabase, Cloud Storage, and API operations for 1,000 users.
- **Personnel**: Maintenance and specialized support/dev.

3. Future Roadmap (Beyond V1)

- **V2**: Multi-format imports (Social media, Notes) and basic trend visualizations.
- **V3**: Advanced narrative editing tools and collaborative library access.
- **V4+**: Live synchronization and API for 3rd party narrative services.

FILE: docs/01_PRODUCT/REQUIREMENTS/ACCESSIBILITY_REQUIREMENTS.md

Accessibility Requirements — Memoir.ai

Memoir.ai is committed to providing an inclusive experience. The following requirements ensure the application is accessible to all users.

1. Visual Accessibility

- **AC-1.1**: **Contrast Ratio**: All text and meaningful UI elements must meet WCAG 2.1 AA standards (minimum 4.5:1 for normal text).
- **AC-1.2**: **Screen Reader Support**: All interactive components must have appropriate ARIA labels and roles.
- **AC-1.3**: **Color Independence**: Information must not be conveyed by color alone (e.g., error states should have icons or text labels).

2. Interaction Accessibility

- **AC-2.1**: **Keyboard Navigation**: Every feature, including search and settings, must be fully navigable via keyboard (Tab, Enter, Arrow keys).
- **AC-2.2**: **Focus Management**: Focus states must be clearly visible and follow a logical order.
- **AC-2.3**: **Input Delays**: Touch and click targets must be at least 44x44 pixels to accommodate users with motor impairments.

3. Cognitive Accessibility

- **AC-3.1**: **Simple Language**: Error messages and guides should avoid technical jargon.
- **AC-3.2**: **Consistent UI**: Navigation patterns must remain stable across all 7 core screens.
- **AC-3.3**: **Calm Motion**: Transitions should be subtle opacity fades to avoid triggering photosensitivity.

4. Verification

Accessibility will be verified using:

- **Automated Audits**: Pa11y or similar tools integrated into the CI/CD pipeline.
- **Manual Testing**: Screen reader (VoiceOver/NVDA) walkthroughs for core user flows.

FILE: docs/01_PRODUCT/REQUIREMENTS/DATA_INTEGRITY_REQUIREMENTS.md

Data Integrity Requirements — Memoir.ai

To ensure that your digital history remains accurate and untampered with, Memoir.ai enforces strict data integrity standards.

1. Immutable Provenance

- **DIR-1.1**: Every `Event` record must include a reference to its original `DataSource` and the specific `ImportJobId`.
- **DIR-1.2**: Once an event is normalized and stored, its core attributes (timestamp, payload, participants) must be treated as immutable.

2. Integrity Hashing

- **DIR-2.1**: The system MUST generate a SHA-256 hash for every raw payload imported.
- **DIR-2.2**: The system MUST generate a "Normalized Hash" for the canonical event record to detect post-ingestion tampering.
- **DIR-2.3**: Media files must be indexed by their SHA-256 content hash to prevent duplicates and verify file health.

3. Verification & Repair

- **DIR-3.1**: The system MUST perform a "Background Health Check" weekly (or upon user request) to verify cross-references between events and media.
- **DIR-3.2**: If an integrity failure is detected (e.g., hash mismatch), the system MUST mark the record as `CORRUPTED` and prompt the user to repair the vault or re-import the source.

4. AI-Specific Integrity

- **DIR-4.1**: AI-generated snapshots must link to the exact IDs of the events used as evidence.
- **DIR-4.2**: If the source evidence for a narrative is deleted from the vault, the narrative must be flagged as `ORPHANED_EVIDENCE`.

FILE: docs/01_PRODUCT/REQUIREMENTS/FUNCTIONAL_REQUIREMENTS.md

Functional Requirements — Memoir.ai

This document defines the core functional capabilities required for the V1 release of Memoir.ai.

1. Vault Management

- **FR-1.1**: The system MUST allow users to create an encrypted vault at a user-specified local directory.
- **FR-1.2**: The system MUST require a user-defined passphrase for vault encryption.
- **FR-1.3**: The system MUST allow users to change their vault passphrase (requires re-encryption of the master key).

2. Data Ingestion (Imports)

- **FR-2.1**: The system MUST ingest and parse iMessage (`chat.db`) archives.
- **FR-2.2**: The system MUST ingest and parse WhatsApp ZIP archives.
- **FR-2.3**: The system MUST extract and store media attachments (images, voice notes) linked to events.
- **FR-2.4**: The system MUST validate archive formats before initiating the import job.

3. Timeline & Search

- **FR-3.1**: The system MUST display all ingested events in a unified, chronological feed.
- **FR-3.2**: The system MUST support keyword-based search across all `eventData` payloads.
- **FR-3.3**: The system MUST support filtering by data source type and participant name.
- **FR-3.4**: The system MUST reconstruct conversation threads across multiple platforms if participants align.

4. AI & Narratives

- **FR-4.1**: The system MUST generate "Snapshots" (summaries) for a user-selected time range.
- **FR-4.2**: The system MUST include verifiable citations (links to source evidence) in all AI-generated outputs.
- **FR-4.3**: The system MUST allow users to edit and save narrative drafts.

5. Security & Privacy

- **FR-5.1**: The system MUST ensure all data processing (parsing, AI inference) occurs locally on the user's device.
- **FR-5.2**: The system MUST provide a "Zero Telemetry" mode, preventing all outbound network requests.

- **FR-5.3**: The system MUST allow for a complete "Vault Deletion" which wipes all encrypted data and indices.

FILE: docs/01_PRODUCT/REQUIREMENTS/NON_FUNCTIONAL_REQUIREMENTS.md

Non-Functional Requirements — Memoir.ai

This document defines the quality attributes, performance targets, and technical constraints for Memoir.ai.

1. Performance

- **NFR-1.1**: **Search Latency**: Keyword search queries must return results in < 200ms on a database of 50,000 events.
- **NFR-1.2**: **Timeline Smoothness**: The infinite scroll must maintain 60 FPS during rapid navigation.
- **NFR-1.3**: **Import Throughput**: The system must process at least 100 messages per second during the parsing phase.

2. Security & Privacy

- **NFR-2.1**: **Encryption Standard**: All data must be encrypted with AES-256 via SQLCipher.
- **NFR-2.2**: **Local Sovereignty**: Zero raw user data may be transmitted to external servers.
- **NFR-2.3**: **Memory Sanitization**: Passphrases and master keys must be cleared from memory immediately after vault unlock/lock.

3. Reliability

- **NFR-3.1**: **Data Integrity**: The system must verify checksums for all media and database records during a health check.
- **NFR-3.2**: **Resume-ability**: Import jobs must be resumable after an application crash or system restart.

4. Usability

- **NFR-4.1**: **Onboarding Time**: A first-time user should be able to create a vault and start their first import in < 3 minutes.
- **NFR-4.2**: **Design Consistency**: The UI must strictly adhere to the "Nebula" cinematic design system.

5. Portability

- **NFR-5.1**: **OS Support**: The application must run natively on macOS (Intel/Silicon) and Windows 10/11.

- **NFR-5.2**: **Export Format**: All data must be exportable in open, non-proprietary formats (JSON/Markdown).

FILE: docs/01_PRODUCT/USER_FLOWS/FLOW_0001_CREATE_LIBRARY.md

User Flow: Create / Open Library (FLOW_0001)

This flow describes the initial setup of the Memoir.ai vault.

1. Flow Diagram

```
graph TD
    A[Launch App] --> B{Vault Exists?}
    B -- No --> C[Onboarding / Philosopy]
    C --> D[Choose Storage Folder]
    D --> E[Create Passphrase]
    E --> F[Initialize SQLCipher DB]
    F --> G[Direct to Import Wizard]
    B -- Yes --> H[Prompt for Passphrase]
    H --> I[Validate Key]
    I -- Success --> J[Load Unified Timeline]
    I -- Fail --> H
```

2. Narrative Description

1. **Launch**: The user opens the Electron application.
2. **Detection**: The app checks for a localized configuration file or default vault path.
3. **Onboarding**: New users see the "Nebula" cinematic introduction explaining local-first privacy.
4. **Creation**: User selects a local directory (e.g., `~/Documents/MemoirVault`).
5. **Security**: User enters a passphrase. A strength meter provides real-time feedback.
6. **Initialization**: The backend runs `PRAGMA key = 'passphrase'` on a new SQLite file and creates the canonical schema.

3. Edge Cases

- **Permission Denied**: If the storage folder is read-only, show a recovery error prompting for a new location.
- **Lost Passphrase**: Display a clear warning that data is unrecoverable without the key.

FILE: docs/01_PRODUCT/USER_FLOWS/FLOW_0002_IMPORT_SOURCE.md

User Flow: Import Data Source (FLOW_0002)

This flow describes bringing raw digital archives into the encrypted vault.

1. Flow Diagram

```
graph TD
    A[Open Import Wizard] --> B[Select Source Type]
    B --> C[Choose File / Folder]
    C --> D[Local Validation]
    D -- Success --> E[Review Import Plan]
    E --> F[Initiate Job]
    F --> G[Job Runner: Parsing]
    G --> H[Job Runner: Indexing]
    H --> I[Final Health Check]
    I --> J[Update Timeline]
```

2. Narrative Description

1. **Selection**: User chooses from supported sources (iMessage, WhatsApp, JSON export).
2. **Point**: User selects the local path to the archive (e.g., `~/Library/Messages/chat.db`).
3. **Validation**: The system checks if the file is readable and matches the expected parser schema.
4. **Planning**: A summary screen estimates event count and storage impact.
5. **Execution**: The **Job Runner** handles the heavy lifting in background threads, showing progress bars for each phase.

3. Edge Cases

- **Corrupt Archive**: If the parser encounters a malformed record, it logs a non-fatal error and continues.
- **Disk Full**: The import pauses and notifies the user if storage limits are hit.

FILE: docs/01_PRODUCT/USER_FLOWS/FLOW_0003_TIMELINE_BROWSE_SEARCH.md

User Flow: Timeline Browsing & Search (FLOW_0003)

This flow describes navigating and querying the unified digital history.

1. Flow Diagram

```
graph TD
    A[Open Unified Timeline] --> B[Infinite Scroll Feed]
    B --> C{Search Triggered?}
    C -- Yes --> D[Global Search Bar Overlay]
    D --> E[Enter Keyword / Semantic Query]
    E --> F[Filter Results locally]
    F --> G[Update Feed View]
    C -- No --> H[Apply Meta Facets]
    H --> I[Filter by Source / Person / Date]
    I --> G
```

2. Narrative Description

1. **Feed**: The primary view is a master chronological stream of every event.
2. **Query**: Users type in the persistent Topbar search. The backend executes a parameterized FTS query against the SQLCipher DB.
3. **Facets**: Users can toggle sources (e.g., "Show only WhatsApp") or filter by specific people identified in communications.
4. **Navigation**: Users can "Date Jump" using the sidebar to navigate years or decades instantly.

3. Edge Cases

- **No Results**: Show a "Nebula" empty state with suggestions for broader search terms.
- **Large Result Sets**: Pagination/Virtualization ensures the UI remains responsive even for 1,000+ matches.

FILE: docs/01_PRODUCT/USER_FLOWS/FLOW_0004_SNAPSHOT_GENERATE_REVIEW.md

User Flow: Snapshot Generation & Review (FLOW_0004)

This flow describes the local AI process for creating citation-backed summaries.

1. Flow Diagram

```
graph TD
    A[Select Events / Time Range] --> B[Open Snapshot Lab]
    B --> C[Configure Tone / Length]
    C --> D[Local AI inference]
    D --> E[Guardrail Validation]
    E -- Pass --> F[Preview with Citations]
    E -- Fail --> G[Unverified Flag / Error]
    F --> H[User Review / Edit]
    H --> I[Save to Library]
```

2. Narrative Description

1. **Scope**: User highlights a group of messages or selects "Year in Review".
2. **Configuration**: User chooses a tone (Neutral, Reflective) and length.
3. **Generation**: The local AI engine processes the normalized data.
4. **Verification**: **Hallucination Guards** verify that every factual claim aligns with a stored evidence ID.
5. **Review**: The user views the draft. Clicking a citation opens a sidebar showing the original message/event.

3. Edge Cases

- **Insufficient Data**: If the range has < 5 events, the AI suggests a broader selection.
- **Hallucination Detected**: The specific sentence is highlighted as `UNVERIFIED` for manual correction.

FILE: docs/01_PRODUCT/USER_FLOWS/FLOW_0005_EXPORT_DELETE.md

User Flow: Export & Deletion (FLOW_0005)

This flow describes the "Local Sovereignty" features for data portability and privacy.

1. Flow Diagram

```
graph TD
    A[Open Settings > Vault] --> B{Select Action}
    B -- Export --> C[Select Format: JSON / Markdown]
    C --> D[Select Scope: All / Filtered]
    D --> E[Generate Portable Archive]
    E --> F[Download Locally]
    B -- Delete --> G[Vault Wipe Confirmation]
    G --> H[Requires Passphrase]
    H -- Correct --> I[Secure Erasure of Database & Files]
    I --> J[Return to Setup Screen]
```

2. Narrative Description

1. **Access**: User navigates to the Vault section in Settings.
2. **Export**: To ensure no lock-in, users can dump their entire normalized history into human-readable JSON/MD files.
3. **Deletion**: A destructive flow that requires a passphrase to prevent accidental wipes.
4. **Wipe**: The application deletes the `.sqlite` file and clears the media cache folder.

3. Edge Cases

- **Partial Export**: If the vault is very large, the export is packaged into multiple ZIP volumes.
- **Incorrect Passphrase**: Deletion is blocked until the identity is verified via the vault key.

TECHNICAL AND DATA SPECIFICATION

Generated on: 2026-01-30T16:22:34.958Z

FILE: docs/03_ARCHITECTURE/DATA_FLOW_DIAGRAMS.md

Data Flow Diagrams

High-level visualization of how bits move through Memoir.ai.

1. Ingestion Flow

```
graph LR
    A[Raw Archive] --> B(Extraction Worker)
    B --> C{Validator}
    C -->|Valid| D[Event Normalizer]
    D --> E[(SQLCipher DB)]
    C -->|Invalid| F[Error Log]
```

2. AI Snapshot Flow

```
graph TD
    A[(SQLCipher DB)] --> B[Context Sampler]
    B --> C[Local LLM Inference]
    C --> D[Citation Engine]
    D --> E[Factual Verifier]
    E --> F[Memoir Narrative]
```

3. Sync Flow (Non-Vault)

```
graph LR
    A[App Local] --> B{Subscription Check}
    B -->|Pro| C[Supabase Auth]
    C --> D[Stripe Status]
```

FILE: docs/03_ARCHITECTURE/MODULE_MAP.md

Module Map — Memoir.ai

This document maps the logical modules of the Memoir.ai application and their boundaries, as defined in the technical specification.

1. System Components

Frontend (Desktop Client)

- **Env**: Electron (Shell), React (UI Layer).
- **State Management**: Redux (Global Application State).
- **Modules**:
  - **Auth Module**: Handles registration, login, and JWT persistence.
  - **Workspace Module**: Manages the Unified Timeline and Conversation Viewer.
  - **Search Module**: Interfaces with the backend for semantic and keyword queries.
  - **Settings Module**: User profile management and data export controls.

Backend (Local Core)

- **Env**: Node.js, Express.js.
- **Database**: SQLite with SQLCipher (Encrypted Persistence).
- **Modules**:
  - **Ingestion Engine**: Handles multi-source archive parsing and normalization.
  - **Job Runner**: Manages background processing of data imports/exports.
  - **Narrative Engine**: Logic for processing events into summaries and drafts.
  - **Security Layer**: Input validation, sanitization, and encryption management.

2. Component Boundaries

Source	Destination	Protocol	Purpose
Electron Shell	**React Frontend**	IPC / Context Bridge	System-level integration (File Dialogs, Windows).
React Frontend	**Node.js Backend**	RESTful APIs (Local)	Data retrieval, Job initiation, Auth requests.
Node.js Backend	**SQLite DB**	SQL (SQLCipher)	Encrypted data persistence.
Node.js Backend	**Local Filesystem**	FS API	Raw file ingestion and ZIP export creation.

3. Communication Patterns

- **Request-Response**: Primary pattern for UI interactions (e.g., fetching a timeline range).
- **Asynchronous Jobs**: Background tasks (Parsing, Indexing) reported via progress updates.
- **Event-Driven**: Progress indicators in the frontend responding to job runner state changes.

FILE: docs/03_ARCHITECTURE/SYSTEM_OVERVIEW.md

System Architecture Overview

The structural blueprint of the Memoir.ai "Fortress."

1. Multi-Process Model (Electron)

- **Main Process**: Handles IPC, File System access, and Window lifecycle.
- **Renderer Process**: The "Nebula" UI (React).
- **Worker Processes**: Low-priority background tasks for heavy compute (Ingestion, AI).

2. Storage Layer

- **Vault Partition**: The encrypted SQLCipher database + Media subfolders.
- **Config Partition**: Unencrypted app settings (Theme, Window position).

3. Implementation Stack

See `docs/00_META/ADR/ADR_0001_STACK_DECISION.md` for full breakdown.

4. Security Philosophy

- **Process Isolation**: The Renderer has no direct access to `fs`. All data requests go through an IPC bridge with strict validation.
- **Ephemeral Keys**: Encryption keys are only held in memory for the duration of the session.

FILE: docs/03_ARCHITECTURE/JOBS_WORKERS/IMPORT_PIPELINE.md

Import Pipeline Specification — Memoir.ai V1

1. Goal

Provide robust, resumable, and transparent background processing for a single message export format (e.g., CSV/JSON).

2. Pipeline Stages

Step	Component	Action	Output
1. Initiate	Frontend/API	Create `import_jobs` record, upload	`import_jobs` record, file in Storage.

Step	Component	Action	Output
		file to Storage.	
2. Process	Worker	Download and parse raw file from Storage.	Structured data stream.
3. Normalize	Worker	Transform raw data into canonical `memories` schema.	Canonical Memory objects.
4. Ingest	Worker	Batch insert memories into Postgres (linked to library).	New `memories` records.
5. Finalize	Worker	Update job status to 'complete' or 'failed'.	Finalized `import_jobs` record.

3. Technical Requirements

- **Resumability**: The worker must handle temporary failures and retry from the last successful batch.
- **Logging**: Detailed error and status logs must be stored in the `log_output` (JSONB) column.
- **Progress Tracking**: `progress_percentage` must be updated in real-time for frontend display.
- **File Isolation**: Raw imports must be stored in user-specific storage paths.

FILE: docs/03_ARCHITECTURE/JOBS_WORKERS/JOB_RUNNER_SPEC.md

Job Runner Specification

The engine responsible for all heavy background compute in Memoir.ai.

1. Execution Philosophy

- **Local-only**: All jobs run in separate worker threads within the Electron backend.
- **Priority-based**: High-priority jobs (Search Indexing) preempt low-priority jobs (Media Thumbnailing).
- **Non-blocking**: The UI remaining responsive even during 50k+ record imports.

2. Job Lifecycle

1. **Enqueue**: Task added to SQLite-backed queue.
2. **Dispatch**: Worker thread claims the oldest high-priority task.
3. **Active**: Periodic `progress` events emitted to the UI via IPC.
4. **Completion/Failure**: Results persisted and status updated.

3. Worker Configuration

- **Max Threads**: Configurable (default: CPUs - 1).
- **Memory Throttling**: Jobs pause if application memory usage exceeds 1.5GB to maintain OS stability.
- **Persistence**: Job state is stored in the `import_jobs` table to allow resumption after crash.

4. Specific Runners

- **Parser**: Specialized for `chat.db`, `WhatsApp.zip`, etc.
- **Indexer**: Updates SQLite FTS5 tables and vector stores.
- **AI Narrative**: Manages local LLM inference and verification.

FILE: docs/03_ARCHITECTURE/JOBS_WORKERS/QUEUE_MODEL.md

Queue Model & Data Structures

Defines how background tasks are represented and managed.

1. Database Schema (`job_queue`)

Field	Type	Description
`job_id`	UUID	Primary Key
`type`	String	e.g., `PARSE_IMESSAGE`, `GEN_SNAPSHOT`
`priority`	Integer	0 (Low) to 10 (Critical)
`payload`	JSON	Job parameters (file paths, UID, etc.)
`status`	Enum	`PENDING`, `RUNNING`, `PAUSED`, `FAILED`, `DONE`
`progress`	Float	0.0 to 1.0
`error_log`	Text	Captured stack trace if failed

2. Priority Levels

- **P10 (Urgent)**: Onboarding flow jobs (Initial Vault Setup).
- **P5 (Normal)**: User-triggered imports.
- **P1 (Background)**: Media thumbnailing, long-term integrity scans.

3. Worker Interaction

Workers use a "Pull" model. They query the `job_queue` for the next available `PENDING` task, mark it as `RUNNING` within an atomic transaction to prevent double-claiming, and begin execution.

FILE: docs/03_ARCHITECTURE/JOBS_WORKERS/RETRIES_IDEMPOTENCY.md

Retries & Idempotency Rules

Ensuring data integrity during failure recovery.

1. Retry Semantic

- **Transient Failures**: (e.g., File system lock) Automatic retry with exponential backoff.
- **Fatal Failures**: (e.g., Disk Full, Permission Denied) Stop and notify user for manual intervention.
- **Max Retries**: Default 3 attempts per job chunk.

2. Idempotency Requirements

- **Parser Idempotency**: Running the same iMessage import twice MUST NOT result in duplicate events. The parser must check the `source_record_id` and `payload_hash` before insertion.
- **Media Ingestion**: Media files are only copied/indexed if their content hash (`SHA256`) does not already exist in the library.

3. Checkpointing

For large imports (100k+ records), the parser must commit batches to the database every 1,000 records. If the job is interrupted, it resumes from the last successfully committed batch ID.

FILE: docs/03_ARCHITECTURE/PROVENANCE_VERSIONING/CITATION_SYSTEM.md

Citation System Specification

Ensuring factual traceability between AI narratives and original source evidence.

1. Citation Anatomy

A citation is represented in the database as a link between a `Narrative` and one or more `Events`.

Attribute	Description
`citation_id`	Unique UUID
`target_narrative_id`	Foreign Key to Narrative
`source_event_ids[]`	Array of Event IDs used as supporting evidence
`anchor_text`	The specific sentence or phrase in the draft being supported
`relevance_score`	AI-assigned confidence (0.0 - 1.0)

2. UI Representation

In the Snapshot Editor, citations appear as superscript numbers (e.g., `Claim text [1]`).

- **Interaction**: Clicking `[1]` opens the "Evidence Drawer", displaying the raw message content, timestamp, and source metadata for the linked events.

3. Integrity Rules

- If a source `Event` is deleted, its linked `Citations` are invalidated.
- The UI MUST visually flag narratives with orphaned citations as `UNVERIFIED`.

FILE: docs/03_ARCHITECTURE/PROVENANCE_VERSIONING/PROVENANCE_MODEL.md

Provenance Model

The chain of custody for every memory record in Memoir.ai.

1. Provenance Meta-Data

Every `Event` record includes an immutable `provenance` block:

- **`origin_device`**: Local hostname where the import occurred.
- **`parser_signature`**: The version of the parser code used (e.g., `imessage-parser-v1.4.2`).
- **`raw_source_path`**: Relative path to the file within the imported archive (e.g., `chat.db/message/123`).
- **`ingestion_timestamp`**: When the record was first created in the vault.

2. Verification

During a "Deep Health Check", the system attempts to re-locate the raw record (if the source archive is still present) and verify its `payload_hash` against the current database state.

3. Trust Levels

- **Level 3 (High)**: Directly imported from a platform DB (e.g., `chat.db`).
- **Level 2 (Medium)**: Imported from a flattened JSON/CSV export.
- **Level 1 (Low)**: Manually added or manually edited by the user.

FILE: docs/03_ARCHITECTURE/PROVENANCE_VERSIONING/SNAPSHOT_SYSTEM.md

AI Snapshot System (Citations & Versioning) — Memoir.ai V1

1. AI Snapshot Generation

The system uses an LLM to transform selected memories into a cohesive narrative summary.

Flow

1. **Input Selection**: User selects a set of `memory_ids` or a time range on the timeline.
2. **Prompt Construction**: The backend worker fetches the selected memories and prompts the LLM to generate a summary.
3. **Citation Mapping**: The LLM is instructed to include inline markers (e.g., `[1]`) that map to specific input memories.
4. **Record Creation**:
   - Creates a new `ai_snapshots` (metadata) record if it's the first in a thread.
   - Creates an `ai_snapshot_versions` record for the narrative content.
   - Creates `snapshot_citations` records linking markers to source `memory_ids`.

2. Data Provenance (Citations)

- **Markers**: Snapshot text contains clickable markers.
- **Source Linking**: Interaction with a marker highlights or navigates the user to the exact source Memory in the timeline.
- **Integrity**: Citations are non-destructive and persistent across versions.

3. Versioning Strategy

- Every "Save" or "Regenerate" operation creates a **new version record**.
- The UI defaults to the highest `version_number` but allows browsing and reverting to historical versions.
- This approach ensures user edits do not destroy the original AI output or previous manual drafts.

FILE: docs/03_ARCHITECTURE/PROVENANCE_VERSIONING/SNAPSHOT_VERSIONING_MODEL.md

Snapshot Versioning Model

Managing the evolution of AI-generated narratives.

1. Revision Strategy

Narratives in Memoir.ai are versioned using a simple linear increment (V1, V2, V3).

2. Version Storage

Instead of overwriting, every "Regenerate" operation creates a NEW record in the `narrative_versions` table.

- **Parent ID**: Link to the root Narrative object.
- **Diff**: The system stores the full content for each version (simpler for local AI consumption) rather than incremental diffs.

3. UI Navigation

The "Version Browser" in the Snapshot Editor allows users to:

- **Compare**: Side-by-side view of V[X] and V[Current].
- **Reactivate**: Promote an old version to current.
- **Delete**: Remove bad drafts to save metadata space.

4. Stability Rule

Once a Narrative is "Published" (exported or pinned), the current version is locked. Future changes must create an explicit new version.

FILE: docs/03_ARCHITECTURE/SEARCH/INDEXING_STRATEGY.md

Indexing Strategy — Memoir.ai

Indexing is the final stage of the Memoir.ai Ingestion Pipeline, ensuring that raw digital history is transformed into a highly performant, queryable dataset.

1. Indexing Workflow

When a parsing job completes, the system initiates indexing:

1. **Normalization**: Raw data is mapped to the `Events` canonical schema.
2. **Schema-on-Read Optimization**: Common fields are indexed; custom `eventData` JSON fields are made searchable.
3. **Health Check**: Indices are verified for integrity before directing the user to the timeline.

2. Technical Choices

- **Database**: SQLite with SQLCipher.
- **Indices**:
  - B-Tree indices on `id`, `userId`, `dataSourceId`, and `createdAt`.
  - Full-Text Search (FTS) indices (if applicable via SQLite FTS5) for keyword-heavy `eventData`.
- **Batching**: Indexing runs in background workers during the Job Runner phase to prevent main-thread blocking.

3. Security & Performance

- **Encrypted Indices**: All indices are stored within the encrypted SQLCipher vault, maintaining the local-first security posture.
- **Resumable Jobs**: Indexing jobs are tracked by the Job Runner Dashboard, allowing for recovery if the app is closed during processing.
- **Latency Target**: Search queries must return results in < 200ms, even for libraries with 50,000+ events.

FILE: docs/03_ARCHITECTURE/SEARCH/QUERY_LANGUAGE_SPEC.md

Query Language Specification

Grammar and reserved keywords for searching the Memoir.ai unified timeline.

1. Syntax Overview

Memoir.ai uses a simplified SQL-like DSL for advanced multi-parameter queries, which are translated into SQLite FTS5 queries.

2. Keywords & Facets

Facet	Description	Example
`from:`	Filter by sender/author	`from: "Mom"`
`to:`	Filter by recipient	`to: "Dad"`
`source:`	Filter by platform	`source: imessage`
`date:`	Specific date or range	`date: 2023-01-01..2023-12-31`
`type:`	Event category	`type: media` or `type: text`
`has:`	Content attributes	`has: link` or `has: attachment`

3. Logical Operators

- **AND**: Implicit (e.g., `from: Mom source: whatsapp`)
- **OR**: Explicit (e.g., `from: Mom OR from: Dad`)
- **NOT**: Prefix with - (e.g., `source: whatsapp -has: link`)

4. Semantic Search

Prefixing a query with ~ triggers a semantic/vector search instead of a keyword match.

- Example: ~ `"When was the last time we talked about the trip?"`

5. Implementation Note

The frontend input component parses these tokens and highlights them for the user. The backend then constructs the `WHERE` clause dynamically, ensuring all inputs are sanitized to prevent SQL injection.

FILE: docs/03_ARCHITECTURE/SEARCH/SEARCH_OVERVIEW.md

Search Subsystem Overview

1. Hybrid Architecture

Combines Lexical (SQLite FTS5) and Semantic (Vector) search.

2. Relevancy Scoring

Final score = `(0.4 * LexicalScore) + (0.6 * SemanticScore)`.

3. Privacy

Vector embeddings are computed locally and stored in the encrypted vault.

FILE: docs/03_ARCHITECTURE/PERFORMANCE/CACHING_STRATEGY.md

Caching Strategy

Optimizing the "Private Time Machine" for local performance.

1. UI State Caching

- **Redux Persist**: Stores UI preferences but NOT user data (to avoid unencrypted spill).
- **Component Memoization**: Heavy use of `React.memo` for the Timeline feed entries.

2. Database Caching

- **SQLite Page Cache**: Configured to 128MB to keep hot indices in memory.
- **Prepared Statements**: Re-used for recurring search filters to avoid parse overhead.

3. Media Caching

- **Thumbnail generation**: During import, the app generates low-res WebP thumbnails stored in the `vault/.thumb/` directory.
- **Action**: The UI displays thumbnails while loading full-res assets from the encrypted blob store.

FILE: docs/03_ARCHITECTURE/PERFORMANCE/LARGE_IMPORTS_STRATEGY.md

Large Imports Strategy

Handling 10GB+ archives without application degradation.

1. The "Chunking" Pattern

The ingestion engine never reads a full file into memory. It uses Node.js streams to process records in chunks of 500.

2. Resource Throttling

- **CPU**: Workers are restricted to `N-1` cores to ensure the Main process remains responsive for UI input.
- **Priority**: Ingestion jobs are marked with `IDLE_PRIORITY` so they yield to real-time AI generation or search requests.

3. Interruption Recovery

- **Checkpointing**: After every 1,000 records, the progress is committed to SQLite.
- **Action**: If the app crashes or the computer shuts down, the import resumes from the last successful checkpoint.

FILE: docs/03_ARCHITECTURE/PERFORMANCE/PERF_BUDGETS.md

Performance Budgets

SLA-style targets for local interaction.

Metric	Budget (P95)	Context
Vault Unlock	< 1.0s	PBKDF2 Hashing -> DB Open
Timeline Scroll	60 FPS	Jitter-free vertical movement
Global Search	< 250ms	Keyword search across 50k events
Semantic Search	< 1.5s	Vector embedding -> Lookup
Snapshot Draft	< 8.0s	Model inference for 1

Metric	Budget (P95)	Context
		chapter
Cold Start	< 2.0s	App launch -> Splash screen

FILE: docs/04_DATA/CANONICAL_DATA_MODEL.md

Canonical Data Model

The unified schema for all personal history.

1. The `Event` Entity

The core atom of Memoir.ai.

Field	Type	Description
`event_id`	UUID	Primary Key
`source_id`	UUID	FK to the data source
`timestamp`	ISO8601	Precision to milliseconds
`platform`	Enum	iMessage, WhatsApp, Email, etc.
`content_raw`	Text	Raw unformatted body
`metadata`	JSONB	Platform-specific headers

2. The `Person` Entity

Extracted participants.

Field	Type	Description
`person_id`	UUID	Primary Key
`display_name`	String	User-provided or extracted
`identities`	JSONB	Map of phone numbers/emails

3. Relationships

- **Event -> Person**: Many-to-Many via `participants` table.
- **Event -> Snapshot**: Linked via `citations`.

FILE: docs/04_DATA/ENTITY_DICTIONARY.md

Entity Dictionary

Detailed semantic definitions for project terminology.

Core Entities

- **DataSource**: A connection to a remote/local archive (e.g., "Troy's iPhone iMessage").
- **Attachment**: A binary file (Image, Video, Audio) linked to an Event.
- **Thread**: A chronological cluster of messages between a set of participants.
- **NarrativeBlock**: A single paragraph or section of an AI Snapshot.

Field Semantics

- `normalized_at`: The UTC time when the record was finalized in the vault.
- `entropy_score`: Internal metric for the information density of a message (used by AI sampler).
- `citation_anchor`: The exact range of characters in a narrative linked to a source.

FILE: docs/04_DATA/DATA_VALIDATION/DEDUPE_MERGE_RULES.md

Deduplication & Merge Rules

Managing overlapping records from multiple data sources.

1. Deduplication (Exact Match)

Records are merged if they meet ALL of the following criteria:

- **Timestamp**: Identical within ±1 second.
- **Participants**: Exact match of resolved entity IDs.
- **Payload Hash**: SHA-256 of the normalized text content is identical.

2. Fuzzy Merging (Overlapping Events)

If records originate from different sources (e.g., an iMessage and a WhatsApp export) but describe the same event:

- **Rule**: If the timestamp is within ±30 seconds and the text similarity is > 90%, group them as a "Consolidated Event".
- **Metadata**: Retain BOTH `source_ids` and `raw_payloads` specifically for provenance verification.

3. Media Deduplication

- **Rule**: All media is indexed by SHA-256 content hash.
- **Action**: If a photo is sent in iMessage and later imported from a local folder, it is stored only ONCE. The database creates two `event_media` links pointing to the same media entry.

4. Conflict Resolution

If two sources provide conflicting metadata (e.g., different "Read" timestamps):

- **Priority**: Higher-integrity databases (e.g., live `chat.db`) override flattened ZIP exports.
- **Conservative Approach**: If priority is equal, keep BOTH values under a `metadata_conflicts` field.

FILE: docs/04_DATA/DATA_VALIDATION/IMPORT_VALIDATION_RULES.md

Import Validation Rules

Strict quality gates for data entering the Memoir.ai vault.

1. Schema Validation

Every record must conform to the `Event` schema before it is inserted into the encrypted datastore.

- **Required Fields**: `event_id`, `timestamp_utc`, `source_id`, `event_type`.
- **Constraint**: `timestamp_utc` must be a valid ISO 8601 string or Unix Epoch.

2. Source Integrity Checks

- **MBOX**: Must have valid headers (`From:`, `Date:`) and a non-empty body.
- **SQLite (iMessage/WhatsApp)**: The table schema must match the expected version for that platform's parser.
- **JSON Exports**: Must conform to the specific platform's current export structure (e.g., Instagram v3 Export).

3. Data Cleansing

- Remove `NULL` characters or malformed byte sequences that could crash indexers.
- Trim whitespace from participant names.
- Resolve relative media paths to absolute paths within the temporary import sandbox.

4. Rejection Policy

If > 10% of a batch fails structural validation, the entire `ImportJob` is paused, and the user is prompted for a "Deep Scan" or "Repair" of the source file.

FILE: docs/04_DATA/DATA_VALIDATION/NORMALIZATION_RULES.md

Normalization Rules

Transforming diverse data formats into the "Memoir.ai Canonical Schema".

1. Participant Normalization

- **Mapping**: Map platform-specific handles (e.g., `+123456789`, `user_123`, `mom@email.com`) to a single `person_id`.
- **Primary Name**: Use the richest metadata available (e.g., Apple Contacts name overrides raw phone number).

2. Timestamp Standardization

- All internal storage is in **UTC**.
- The `UI_DISPLAY_TIME` is calculated on the fly using the `vault_timezone` stored in the user profile.

3. Event Type Mapping

Raw Source Type	Canonical `event_type`
iMessage SMS / WhatsApp Chat	`MESSAGE`
Instagram Post / FB Wall	`SOCIAL_POST`
Gmail / Outlook	`EMAIL`
JPG / PNG (No message)	`PHOTO_EVENT`

4. Attachment Handling

- Attachments are extracted and assigned a unique `media_id`.
- If an event is a "Group Chat", the normalization layer creates links to all participants in the `event_participants` junction table.

FILE: docs/04_DATA/EXPORT_FORMATS/EXPORT_SCHEMA.md

Export Schema (JSON)

Ensuring data portability without proprietary lock-in.

1. Export Scope

The export includes all normalized events, narratives, and participant data in a machine-readable format.

2. Event Structure

```json
{
  "export_metadata": {
    "vault_id": "uuid",
    "exported_at": "timestamp",
    "version": "1.0.0"
  },
  "events": [
    {
      "id": "event_uuid",
      "timestamp": "2023-01-01T12:00:00Z",
      "type": "MESSAGE",
      "source": "imessage",
      "author": "Alice",
      "content": "Hello world!",
      "attachments": ["media_id_1.jpg"],
      "provenance": "chat.db/message/123"
    }
  ]
}
```

3. Narrative Structure

AI drafts include citations linked back to event IDs within the same export bundle.

```json
{
  "narratives": [
    {
      "id": "narrative_uuid",
      "title": "Summer Trip",
      "body": "We went to the beach [1].",
      "citations": [
        { "id": 1, "evidence_ids": ["event_uuid_456"] }
      ]
    }
  ]
}
```

FILE: docs/04_DATA/EXPORT_FORMATS/EXPORT_ZIP_LAYOUT.md

Export ZIP Layout

Organization of the human-readable data package.

1. Directory Structure

```
Memoir_Export_[Date]/
├── data/
│   ├── events.json
│   ├── narratives.json
│   ├── library_metadata.json
├── media/
│   ├── images/
│   ├── videos/
│   └── originals/
├── docs/
│   ├── Narratives_Markdown/ (User-readble mirror)
│   └── README_EXPORT_MAP.md
└── LICENSE_DATA_SOVEREIGNTY.txt
```

2. Media Organization

- Media is stored in the `media/` folder, indexed by their `SHA256` hash (e.g., `media/images/a1b2c3d4...jpg`).
- The `events.json` file uses these relative paths for easy reconstruction by external tools.

3. Narrative Mirroring

To ensure users can read their memoirs without special software, every `Narrative` is also exported as a standalone `.md` file in `docs/Narratives_Markdown/`.

4. Integrity

The root level contains a `checksums.txt` file listing the MD5/SHA256 hashes of every file in the export bundle.

FILE: docs/04_DATA/EXPORT_FORMATS/JSON_SCHEMAS/provenance.schema.json

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Provenance Export Schema",
  "type": "object",
  "properties": {
    "export_id": { "type": "string", "format": "uuid" },
    "citations": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "source_event_id": { "type": "string" },
          "target_fragment": { "type": "string" },
          "confidence_score": { "type": "number" }
        }
      }
    }
  }
}
```

FILE: docs/04_DATA/EXPORT_FORMATS/JSON_SCHEMAS/snapshot.schema.json

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Snapshot Export Schema",
    "type": "object",
    "properties": {
        "snapshot_id": {
            "type": "string"
        },
        "version": {
            "type": "integer"
        },
        "narrative_body_markdown": {
            "type": "string"
        },
        "generated_at": {
            "type": "string",
            "format": "date-time"
        }
    }
}
```

FILE: docs/04_DATA/EXPORT_FORMATS/JSON_SCHEMAS/timeline.schema.json

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Timeline Export Schema",
    "type": "object",
    "properties": {
        "timeline_id": {
            "type": "string"
        },
        "entries": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/entry"
            }
        }
    },
    "definitions": {
        "entry": {
            "type": "object",
            "properties": {
                "event_id": {
                    "type": "string"
                },
                "timestamp": {
                    "type": "string"
                },
                "platform": {
                    "type": "string"
                },
                "content": {
                    "type": "string"
                }
            }
        }
    }
}
```

FILE: docs/04_DATA/SUPABASE/SCHEMAS/00_SCHEMA_OVERVIEW.md

Schema Overview

High-level map of the Supabase backend tables.

1. The Metadata Hub

While the "Private Bits" live in the Vault, the Supabase backend tracks:

- Subscription visibility.
- Sync health.
- Cross-device metadata (e.g., "Troy's MacBook" vs "Troy's PC").

2. Entity Map

See the individual `.sql` files in this directory for the full relational schema.

- **Auth Layer**: `01_AUTH_WORKSPACES.sql`
- **Job Layer**: `02_SOURCES_IMPORT_JOBS.sql`
- **Snapshot Layer**: `06_SNAPSHOTS_VERSIONS.sql`

FILE: docs/04_DATA/SUPABASE/SCHEMAS/01_AUTH_WORKSPACES.sql

```sql
-- 01_AUTH_WORKSPACES.sql
CREATE TABLE auth_profiles (
    id UUID REFERENCES auth.users PRIMARY KEY,
    full_name TEXT,
    avatar_url TEXT
);

CREATE TABLE workspaces (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    owner_id UUID REFERENCES auth.users,
    slug TEXT UNIQUE,
    settings JSONB DEFAULT '{}'
);
```

FILE: docs/04_DATA/SUPABASE/SCHEMAS/02_SOURCES_IMPORT_JOBS.sql

```sql
-- 02_SOURCES_IMPORT_JOBS.sql
CREATE TABLE data_sources (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID REFERENCES workspaces,
    source_type TEXT NOT NULL,
    config JSONB
);

CREATE TABLE import_jobs (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    source_id UUID REFERENCES data_sources,
    status TEXT,
    error_log TEXT
);
```

FILE: docs/04_DATA/SUPABASE/SCHEMAS/03_PARTICIPANTS_CONVERSATIONS_MESSAGES.sql

```sql
-- 03_PARTICIPANTS_CONVERSATIONS_MESSAGES.sql
CREATE TABLE conversation_threads (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID REFERENCES workspaces,
    platform TEXT,
    external_id TEXT,
    metadata JSONB
);

CREATE TABLE participants (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID REFERENCES workspaces,
    display_name TEXT,
    contact_info JSONB
);

CREATE TABLE message_records (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    thread_id UUID REFERENCES conversation_threads,
    sender_id UUID REFERENCES participants,
    content_text TEXT,
    sent_at TIMESTAMPTZ
);
```

FILE: docs/04_DATA/SUPABASE/SCHEMAS/04_MEDIA_ATTACHMENTS.sql

```
-- 04_MEDIA_ATTACHMENTS.sql
CREATE TABLE media_assets (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID REFERENCES workspaces,
    file_path TEXT,
    mime_type TEXT,
    size_bytes BIGINT,
    metadata JSONB
);

CREATE TABLE message_attachments (
    message_id UUID REFERENCES message_records,
    media_id UUID REFERENCES media_assets,
    PRIMARY KEY (message_id, media_id)
);
```

FILE: docs/04_DATA/SUPABASE/SCHEMAS/05_TIMELINE_ENTRIES.sql

```
-- 05_TIMELINE_ENTRIES.sql
CREATE TABLE timeline_events (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID REFERENCES workspaces,
    event_type TEXT,
    occurred_at TIMESTAMPTZ,
    reference_id UUID, -- Link to message, photo, etc.
    summary_text TEXT
);

CREATE INDEX idx_timeline_workspace_time ON timeline_events
(workspace_id, occurred_at DESC);
```

FILE: docs/04_DATA/SUPABASE/SCHEMAS/06_SNAPSHOTS_VERSIONS.sql

```
-- 06_SNAPSHOTS_VERSIONS.sql
CREATE TABLE snapshot_records (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID REFERENCES workspaces,
    version_id INTEGER DEFAULT 1,
    title TEXT,
    narrative_blob_ref TEXT -- Path in Supabase Storage or UUID
);
```

FILE: docs/04_DATA/SUPABASE/SCHEMAS/07_CITATIONS_PROVENANCE.sql

```
-- 07_CITATIONS_PROVENANCE.sql
CREATE TABLE citations (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    snapshot_id UUID REFERENCES snapshot_records,
    source_event_id UUID,
    confidence_score FLOAT,
    context_snippet TEXT
);
```

FILE: docs/04_DATA/SUPABASE/SCHEMAS/08_TAGS_NOTES_BOOKMARKS.sql

```
-- 08_TAGS_NOTES_BOOKMARKS.sql
CREATE TABLE user_tags (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID REFERENCES workspaces,
    label TEXT,
    color TEXT
);

CREATE TABLE tagged_items (
    tag_id UUID REFERENCES user_tags,
    item_id UUID, -- Link to Event, Participant, etc.
    item_type TEXT,
    PRIMARY KEY (tag_id, item_id)
);
```

FILE: docs/04_DATA/SUPABASE/SCHEMAS/09_EXPORT_DELETE_AUDIT.sql

```sql
-- 09_EXPORT_DELETE_AUDIT.sql
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES auth.users,
    action TEXT,
    metadata JSONB,
    created_at TIMESTAMPTZ DEFAULT now()
);

CREATE TABLE export_history (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID REFERENCES workspaces,
    status TEXT,
    download_url TEXT,
    expired_at TIMESTAMPTZ
);
```

FILE: docs/04_DATA/SUPABASE/RLS_POLICIES/00_RLS_OVERVIEW.md

RLS Overview

Row Level Security (RLS) is the primary defense for multi-tenant isolation in Memoir.ai's cloud metadata layer.

1. Core Principle

- **No Global Reads**: All tables must have RLS enabled.
- **Ownership Check**: Policies MUST verify that `auth.uid() == owner_id` or equivalent.

2. Policy Matrix

Table	SELECT	INSERT/UPDATE
`workspaces`	owner	owner
`import_jobs`	workspace owner	workspace owner
`entitlements`	user	service_role only

3. Enforcement

The Supabase `anon` key is restricted via these policies. The `service_role` key is used only by internal webhooks (e.g., Stripe) to bypass RLS for administrative updates.

FILE: docs/04_DATA/SUPABASE/RLS_POLICIES/RLS_AUDIT_LOGS.sql

```sql
-- RLS for audit logs
ALTER TABLE audit_logs ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users can only view their own audit logs"
ON audit_logs FOR SELECT
USING (auth.uid() = user_id);

CREATE POLICY "System can insert audit logs"
ON audit_logs FOR INSERT
WITH CHECK (true); -- Usually restricted to service_role via API
```

FILE: docs/04_DATA/SUPABASE/RLS_POLICIES/RLS_MEDIA.sql

```sql
-- RLS for media tables
ALTER TABLE media_assets ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users can only access media from their workspaces"
ON media_assets FOR SELECT
USING (auth.uid() IN (
    SELECT owner_id FROM workspaces WHERE id = workspace_id
));

CREATE POLICY "Users can upload media to their workspaces"
ON media_assets FOR INSERT
WITH CHECK (auth.uid() IN (
    SELECT owner_id FROM workspaces WHERE id = workspace_id
));
```

FILE: docs/04_DATA/SUPABASE/RLS_POLICIES/RLS_MESSAGES.sql

```sql
-- RLS for messages (Cloud metadata layer only)
ALTER TABLE cloud_message_metadata ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users can only read their own message metadata"
ON cloud_message_metadata FOR SELECT
```

```sql
USING (auth.uid() IN (
    SELECT owner_id FROM workspaces WHERE id = workspace_id
));
```

FILE: docs/04_DATA/SUPABASE/RLS_POLICIES/RLS_SNAPSHOTS.sql

```sql
-- RLS for snapshots
ALTER TABLE snapshot_records ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Owners manage their snapshots"
ON snapshot_records FOR ALL
USING (auth.uid() IN (
    SELECT owner_id FROM workspaces WHERE id = workspace_id
));
```

FILE: docs/04_DATA/SUPABASE/RLS_POLICIES/RLS_SOURCES_IMPORTS.sql

```sql
-- RLS for sources and imports
ALTER TABLE data_sources ENABLE ROW LEVEL SECURITY;
ALTER TABLE import_jobs ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users manage their own data sources"
ON data_sources FOR ALL
USING (auth.uid() IN (
    SELECT owner_id FROM workspaces WHERE id = workspace_id
));

CREATE POLICY "Users manage their own import jobs"
ON import_jobs FOR ALL
USING (auth.uid() IN (
    SELECT owner_id FROM workspaces WHERE id = workspace_id
));
```

FILE: docs/04_DATA/SUPABASE/RLS_POLICIES/RLS_WORKSPACES.sql

```sql
-- RLS for workspaces
ALTER TABLE workspaces ENABLE ROW LEVEL SECURITY;
```

```sql
CREATE POLICY "Owners can see their workspaces"
ON workspaces FOR SELECT
USING (auth.uid() = owner_id);

CREATE POLICY "Owners can create workspaces"
ON workspaces FOR INSERT
WITH CHECK (auth.uid() = owner_id);
```

FILE: docs/04_DATA/SUPABASE/MIGRATIONS/0001_init.sql

```sql
-- 0001_init.sql
-- Initial migration for Memoir.ai backend.

CREATE TABLE workspaces (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    owner_id UUID REFERENCES auth.users(id),
    name TEXT NOT NULL,
    created_at TIMESTAMPTZ DEFAULT now()
);

CREATE TABLE import_jobs (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID REFERENCES workspaces(id),
    status TEXT CHECK (status IN ('PENDING', 'RUNNING', 'COMPLETED', 
'FAILED')),
    source_type TEXT NOT NULL,
    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMPTZ DEFAULT now()
);

CREATE TABLE entitlements (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES auth.users(id),
    tier TEXT DEFAULT 'FREE',
    stripe_customer_id TEXT,
    updated_at TIMESTAMPTZ DEFAULT now()
);
```

FILE: docs/04_DATA/SUPABASE/MIGRATIONS/0002_add_snapshot_versioning.sql

```sql
-- 0002_add_snapshot_versioning.sql

ALTER TABLE import_jobs ADD COLUMN version_id UUID;
```

```sql
CREATE TABLE snapshot_metadata (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID REFERENCES workspaces(id),
    version_key TEXT NOT NULL,
    checksum TEXT,
    created_at TIMESTAMPTZ DEFAULT now()
);
```

FILE: docs/04_DATA/SUPABASE/MIGRATIONS/0003_add_citations.sql

```sql
-- 0003_add_citations.sql

CREATE TABLE provenance_sync (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID REFERENCES workspaces(id),
    event_id_map JSONB, -- Mapping local Event UUIDs to sync refs
    last_synced_at TIMESTAMPTZ DEFAULT now()
);
```

FILE: docs/04_DATA/SUPABASE/SEEDS/seed_dev_minimal.sql

```sql
-- Seed Dev Minimal
-- Insert test user (Fake UUID)
INSERT INTO auth.users (id, email) VALUES ('f47ac10b-58cc-4372-a567-
0e02b2c3d479', 'test@memoir.ai');

-- Insert initial workspace
INSERT INTO workspaces (owner_id, name) VALUES ('f47ac10b-58cc-4372-
a567-0e02b2c3d479', 'Default Vault');

-- Insert Free Tier entitlement
INSERT INTO entitlements (user_id, tier) VALUES ('f47ac10b-58cc-4372-
a567-0e02b2c3d479', 'FREE');
```

FILE: docs/05_APIS/API_OVERVIEW.md

API Overview

The internal communication layer between the React UI and the Node.js/Electron backend.

1. Protocol

- **IPC (Inter-Process Communication)**: Main communication channel for sensitive vault operations.
- **RESTful Endpoints (Local)**: Used for non-sensitive data retrieval (e.g., UI preferences, non-encrypted metadata).

2. Global Constants

- **Base Local URL**: `http://localhost:port/api/v1`
- **Authentication**: All requests must include the `X-Vault-Token` generated during the unlock flow.

3. Core Controllers

- **/api/vault**: Initialize, Unlock, Lock, Change Passphrase.
- **/api/timeline**: Fetch events, Filter, Search.
- **/api/jobs**: Enqueue import, Monitor progress, Cancel jobs.
- **/api/narratives**: CRUD operations for Snapshots and Drafts.
- **/api/billing**: Subscription status, Stripe session triggers.

4. Response Format

```json
{
  "success": true,
  "data": { ... },
  "error": null,
  "meta": {
    "timestamp": "ISO-8601",
    "requestId": "uuid"
  }
}
```

FILE: docs/05_APIS/AUTHORIZATION_MODEL.md

Authorization Model

Determining access rights within the local-first application.

1. The Master Key

Full access to the vault is strictly gated by the **Passphrase**.

- **Status: Locked**: Only the `/auth` routes are available. All `Event` and `Narrative` requests return `403 Forbidden`.
- **Status: Unlocked**: The backend holds the SQLCipher master key in memory (sanitized on lock). All local API calls are permitted.

2. Role-Based Access Control (RBAC)

While it is an individual-user app, internal RBAC applies to premium feature access:

- **Free**: Restricted imports (max 2 sources), no AI narratives.
- **Pro**: Unlimited sources, full AI Snapshot generation.
- **Team/Shared**: Future capability for multiple IDs within a single multi-participant vault.

3. Entitlement Checks

Every request to the Jobs or AI layers performs a check against the `subscription_status` cached in the secure local preferences.

FILE: docs/05_APIS/ERROR_TAXONOMY.md

Error Taxonomy

Standardized error codes and user-facing recovery messages.

1. Standard Error Object

```
{
  "code": "VAULT_LOCKED",
  "message": "The vault must be unlocked to perform this action.",
  "recovery": "Navigate to the lock screen and enter your 
passphrase.",
  "severity": "FATAL"
}
```

2. Error Categories

Code Prefix	Category	Description
`AUTH_`	Authentication	Passphrase mismatch, session expiry.
`VALT_`	Vault Integrity	Corruption, missing DB file, storage full.
`IMPT_`	Import Engine	Corrupt archive, unknown format, parser crash.
`AI_`	AI Services	Token limit, hallucination guard fail, model offline.
`BILL_`	Commercials	Payment failed, feature gated.

3. Logging Policy

- **UI**: Display the `message` and `recovery` action in a "Nebula Red" toast or modal.
- **Backend logs**: Record the `code` and detailed stack trace locally in `logs/error.log` for troubleshooting.

FILE: docs/05_APIS/ENDPOINTS/API_ENDPOINTS.md

API Endpoints Specification

Public and internal REST endpoints for the Memoir.ai sync service.

1. Auth & Workspaces

- `POST /v1/auth/login`: Exchange Supabase credentials for a session.
- `GET /v1/workspaces`: List all associated vaults for the user.
- `POST /v1/workspaces`: Initialize a new cloud record for a local vault.

2. Billing & Entitlements

- `GET /v1/billing/status`: Fetch current subscription tier and usage counts.
- `POST /v1/billing/checkout`: Generate a Stripe Checkout URL.
- `GET /v1/billing/invoices`: List recent invoice metadata.

3. Sync & Jobs

- `POST /v1/sync/heartbeat`: Update the cloud with local health metrics.
- `GET /v1/jobs/:id/status`: Check the progress of an async cloud verification job.

FILE: docs/05_APIS/ENDPOINTS/BILLING_API.md

Billing API Specification

Endpoints

- `GET /v1/billing/plans`: Fetch available subscription tiers.
- `POST /v1/billing/subscribe`: Initiate Stripe checkout.
- `DELETE /v1/billing/cancel`: Stop recurring payments.

Responses

Standard JSON response with ERROR_TAXONOMY.md codes.

FILE: docs/05_APIS/ENDPOINTS/EXPORT_DELETE_API.md

Export/Delete API Specification

1. Data Export

- `POST /v1/export/request`: Trigger a full vault export.
- `GET /v1/export/status/:id`: Check status of a ZIP generation.
- `GET /v1/export/download/:id`: Get a temporary signed URL for the export.

2. Account Deletion

- `POST /v1/account/delete`: Initiate full account and data purge.
- `POST /v1/account/delete/confirm`: Securely confirm deletion with MFA/Password.

FILE: docs/05_APIS/ENDPOINTS/IMPORTS_API.md

Imports API Specification

1. Import Jobs

- `POST /v1/imports/jobs`: Start a new ingestion process.
- `GET /v1/imports/jobs`: List recent import history.
- `GET /v1/imports/jobs/:id`: Detailed status and error logs.

2. Sources

- `GET /v1/imports/sources`: List configured data connections.
- `DELETE /v1/imports/sources/:id`: Remove a source connection.

FILE: docs/05_APIS/ENDPOINTS/MEDIA_API.md

Media API Specification

1. Asset Management

- `GET /v1/media`: Search media assets with filters.
- `GET /v1/media/:id/url`: Get a temporary URL for high-res viewing.
- `GET /v1/media/:id/thumbnail`: Get low-res preview.

FILE: docs/05_APIS/ENDPOINTS/SEARCH_API.md

Search API Specification

Endpoints

- `POST /v1/search/query`: Submit a hybrid search request.
- `GET /v1/search/history`: Retrieve recent query strings.

Relevancy Parameters

- `alpha`: Balance between Lexical and Semantic.
- `limit`: Max records to return.

FILE: docs/05_APIS/ENDPOINTS/SNAPSHOTS_API.md

Snapshots API Specification

1. Generation

- `POST /v1/snapshots/generate`: Request AI narrative generation.
- `GET /v1/snapshots`: List available snapshots.
- `GET /v1/snapshots/:id`: Retrieve full snapshot body and citations.

2. Iteration

- `PATCH /v1/snapshots/:id`: Save manual edits to a narrative.
- `POST /v1/snapshots/:id/regenerate`: Trigger a new version with updated prompts.

FILE: docs/05_APIS/ENDPOINTS/TIMELINE_API.md

Timeline API Specification

1. Discovery

- `GET /v1/timeline`: Fetch a page of timeline events.
- `GET /v1/timeline/count`: Total event count for progress bars.
- `GET /v1/timeline/stats`: Distribution of events over time/platform.

FILE: docs/05_APIS/EVENT_SCHEMAS/EVENT_SCHEMAS.md

Event Schemas

The shape of messages sent over the IPC and Sync channels.

1. `VAULT_LOCKED` Event

Triggered when the user locks their database or the timeout expires.

```json
{
  "type": "VAULT_LOCKED",
  "payload": {
    "vault_id": "UUID",
    "timestamp": "ISO8601",
    "reason": "USER_ACTION | TIMEOUT"
  }
}
```

2. `IMPORT_PROGRESS` Event

Emitted by the Ingestion Worker.

```json
{
  "type": "IMPORT_PROGRESS",
  "payload": {
    "job_id": "UUID",
    "percent_complete": 45,
    "current_file": "chat.db"
  }
}
```

FILE: docs/05_APIS/EVENT_SCHEMAS/export_ready.event.json

```json
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Export Ready Event Schema",
    "type": "object",
    "properties": {
        "export_id": {
            "type": "string"
        },
        "download_url": {
            "type": "string"
        },
        "expiry": {
            "type": "string",
            "format": "date-time"
        }
    }
}
```

FILE: docs/05_APIS/EVENT_SCHEMAS/import_job.event.json

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Import Job Event Schema",
    "type": "object",
    "properties": {
        "job_id": {
            "type": "string"
        },
        "status": {
            "enum": [
                "PENDING",
                "RUNNING",
                "COMPLETED",
                "FAILED"
            ]
        },
        "records_processed": {
            "type": "integer"
        }
    }
}
```

FILE: docs/05_APIS/EVENT_SCHEMAS/snapshot_generated.event.json

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Snapshot Generated Event Schema",
    "type": "object",
    "properties": {
        "snapshot_id": {
            "type": "string"
        },
        "version": {
            "type": "integer"
        },
        "uri": {
            "type": "string"
        }
    }
}
```

FILE: docs/05_APIS/WEBHOOKS/STRIPE_WEBHOOKS.md

Stripe Webhooks

1. Lifecycle Events

- `checkout.session.completed`: Upgrades user to Pro.
- `customer.subscription.deleted`: Downgrades user to Free.
- `invoice.payment_failed`: Triggers notification flow for payment issues.

2. Security

All webhooks MUST verify the `stripe-signature` header using the signing secret.

FILE: docs/05_APIS/WEBHOOKS/SUPABASE_TRIGGERS.md

Supabase Triggers

1. Database Events

- `on_user_created`: SQL trigger to insert default workspace and free entitlement.
- `on_workspace_deleted`: Cleanup trigger to purge linked metadata.

2. Real-time Listeners

The application listens for `INSERT` on `import_jobs` to refresh the UI progress state.

FILE: docs/05_APIS/WEBHOOKS/WEBHOOKS_SPEC.md

Webhooks Specification

External callbacks handled by the Memoir.ai backend.

1. Stripe Checkout Complete

- **Route**: `POST /webhooks/stripe/checkout-complete`
- **Action**: Update `entitlements` table to reflect new subscription tier.

2. Stripe Invoice Paid

- **Route**: `POST /webhooks/stripe/invoice-paid`

- **Action**: Reset usage meters for the new billing cycle.

3. Supabase Auth Signup

- **Route**: `POST /webhooks/auth/on-signup` (Database Trigger)
- **Action**: Create default "Solo" workspace record.

Memoir.ai — UX_SPECS / SCREEN_SPECS

This document defines implementation-grade UX specifications for screens previously lacking granular detail. It covers component inventories, interaction states, transitions, and routing implications required for engineering execution.

Global UX Conventions

Layout Zones

All screens follow the standard shell:

- Left Sidebar Navigation
- Top Context Bar
- Main Content Workspace
- Optional Right Inspector Panel
- Modal Layer for destructive or blocking flows
- Toast/Notification Layer

Standard UI States

Each screen must support:

- Loading
- Empty
- Error (recoverable)
- Error (fatal)
- Success confirmation
- Background processing indicator
- Offline or locked vault state

Animation Rules ("Nebula" Motion Language)

Animations must remain subtle and calm.

Patterns:

- Soft opacity fades (150–250ms)
- Slight vertical elevation on entry (4–8px)
- Timeline alignment pulse for completed operations
- Non-blocking shimmer for loading content
- Avoid rapid motion or bright flashes

Motion should never delay task completion.

SCREEN: BILLING

Purpose

Manage subscription, payment methods, invoices, and usage limits.

Primary Components

- Plan Summary Card
- Current Tier Indicator
- Upgrade/Downgrade Options
- Usage Metrics Panel
- Payment Method Manager
- Invoice History Table
- Subscription Cancellation Panel
- Trial Status Indicator
- Renewal Countdown Indicator

Interaction Flows

Upgrade Flow

1. User selects plan.
2. Plan comparison modal opens.
3. User confirms selection.
4. Payment method validated.
5. Confirmation displayed.
6. Limits updated live.

Cancellation Flow

1. User clicks cancel subscription.
2. Confirmation modal with retention messaging.
3. User confirms cancellation.
4. Subscription remains active until billing end.
5. UI reflects pending cancellation.

States

Loading:

- Skeleton cards for plans.
- Placeholder invoice list.

Empty:

- No invoices message.
- No payment method warning.

Error:

- Payment failure alert.
- Retry payment action.

Success:

- Toast confirmation after payment update.

SCREEN: SETTINGS

Purpose

Manage vault configuration, privacy, imports, and preferences.

Tabs

1. General
2. Vault & Storage
3. Imports
4. Privacy & Security
5. Snapshots Preferences
6. Advanced Diagnostics
7. Keyboard Shortcuts

Component Breakdown

General:

- Theme toggle
- Default landing screen selector
- Notification preferences

Vault & Storage:

- Vault path display
- Move vault action
- Storage usage visualization
- Export vault data
- Delete vault data

Imports:

- Default import folders
- Auto-deduplication toggle
- Parser performance mode

Privacy & Security:

- Passphrase change flow
- Vault lock timeout setting
- Local analytics toggle

Snapshots Preferences:

- Default narrative tone
- Summary length preference
- Auto-citation toggle

Advanced Diagnostics:

- Logs viewer
- Rebuild index action
- Repair vault action

Keyboard Shortcuts:

- Shortcut viewer
- Custom mapping

States

Loading:

- Tab-level skeleton loaders.

Error:

- Vault path inaccessible warning.
- Index repair suggestion.

Success:

- Toast confirmation after settings save.

SCREEN: SNAPSHOTS

Purpose

Generate, manage, and refine narrative outputs.

Primary Components

- Snapshot List Panel
- Snapshot Preview Panel
- Source Event Selector
- Version History Viewer
- Regenerate Snapshot Button

- Citation Viewer
- Edit Draft Panel
- Export Snapshot Controls

Interaction Flow

Snapshot Creation:

1. User selects events or time range.
2. Snapshot generation job begins.
3. Progress indicator shown.
4. Draft appears in preview.
5. User edits or approves.

Regeneration:

1. User triggers regenerate.
2. Previous version archived.
3. New version replaces preview.

Version Navigation:

- Users can revert to previous versions.
- Version diffs shown inline.

States

Loading:

- Placeholder narrative blocks.

Empty:

- No snapshots created message.

Error:

- Generation failure with retry option.

Success:

- Snapshot created toast confirmation.

ROUTE STRUCTURE

Core Routes

/app /app/timeline /app/imports /app/snapshots /app/settings /app/billing /app/jobs

Nested Routes

/app/snapshots/:snapshotId /app/timeline/event/:eventId /app/settings/:tab

Parameters

- snapshotId
- eventId
- tab
- sourceId

Deep links must restore scroll and selection state.

Accessibility Requirements

- Full keyboard navigation
- Screen reader labeling
- Minimum AA contrast ratios
- Focus states for all interactive elements

Error Handling Philosophy

Errors must always provide:

1. Explanation
2. Recovery action
3. Non-destructive fallback

Never leave user in a blocked state.

Completion Criteria

UX_SPECS considered complete when:

- All screens support loading, empty, error, and success states.
- Navigation routes map to functional screens.
- Motion guidelines consistently applied.
- Settings and billing flows validated.
