AI OPERATIONS AND USER GUIDE

Generated on: 2026-01-30T16:22:34.975Z

FILE: docs/06_AI/AI_OVERVIEW.md

AI Subsystem Overview

1. Local-First Logic

The AI core runs locally using specialized Electron workers to avoid leaking PII (Personally Identifiable Information) to cloud providers.

2. Model Tiering

- **Small (Llama-3-8B)**: Used for on-the-fly categorization and search expansion.
- **Large (Llama-3-70B)**: Used for deep Snapshot generation (requires discrete GPU).

3. Governance

See `docs/06_AI/SAFETY/USER_CONTROL_GUARDRAILS.md` for user override patterns.

FILE: docs/06_AI/SNAPSHOT_PIPELINE/CITATION_INSERTION_RULES.md

Citation Insertion Rules

The logic for embedding evidence within AI narratives.

1. Granularity

Citations MUST be inserted at the **sentence level**.

- **Correct**: "You visited Paris in July [1]."
- **Incorrect**: "You visited Paris in July and saw the Eiffel Tower [1] [2]." (Should be two separate citations if evidence source differs).

2. Formatting

- **Standard**: Superscript bracketed numbers: `[1]`, `[2]`.

- **Media Citations**: If the claim describes an image, the citation includes a small "Photo" icon: `[📸 1]`.

3. Conflict Tagging

If the AI finds two pieces of evidence that contradict each other (e.g., two different dates for the same event):

- **Rule**: The system MUST insert a "Conflict Citation" `[❓ 1, 2]` which, when clicked, shows both conflicting messages side-by-side for user resolution.

FILE: docs/06_AI/SNAPSHOT_PIPELINE/SNAPSHOT_GENERATION_SPEC.md

Snapshot Generation Specification

The on-device execution flow for narrativizing digital history.

1. Trigger & Payload

A generation job is triggered when a user selects a range of `Events`.

- **Payload**:
  - `evidence_set`: JSON array of normalized events.
  - `tone_id`: (clinical | lyrrical | reflective).
  - `max_tokens`: (short | medium | long).

2. Extraction Phase

The system extracts key entities (people, locations) and temporal anchors from the evidence set to build the **Prompt Context**.

3. Inference Phase

- **Model**: Local LLM (e.g., Llama-3-8B-Instruct or similar).
- **Strategy**: One-shot prompting with explicit instructions for citation placement.

4. Post-Processing Phase

1. **Strict Verification**: Run `HALLUCINATION_GUARDS`.
2. **Citation Linking**: Map the `[1]` strings in the LLM output to actual `event_ids`.
3. **Sanitization**: Ensure noforbidden content categories were generated.

FILE: docs/06_AI/SNAPSHOT_PIPELINE/SNAPSHOT_TEMPLATE_LIBRARY.md

Snapshot Template Library

Catalog of pre-defined prompt structures for diverse narrative styles.

1. "The Daily Highlight" (Short)

- **Focus**: Key interactions and visual milestones from a 24-hour period.
- **Tone**: Quick, energetic, observant.

2. "The Relationship Arc" (Medium)

- **Focus**: Communication patterns and emotional shifts between two specific participants over months.
- **Tone**: Analytical, reflective, warm.

3. "The Odyssey" (Long/Chapter)

- **Focus**: Major life events, travel, or project-based groupings.
- **Tone**: Lyrical, detailed, cinematic.

4. "The Digital Silence"

- **Focus**: Identifying gaps in communication and periods of low digital activity for self-reflection.
- **Tone**: Quiet, introspective.

FILE: docs/06_AI/SNAPSHOT_PIPELINE/VERSIONING_RULES.md

Versioning Rules (AI Narratives)

Handling the iteration and evolution of generated history.

1. Revision Logic

- Every "Regenerate" click MUST create a new immutable version in the `narrative_versions` table.
- **Incrementing**: V1 -> V2 -> V3.

2. User Edits

- If a user manually edits an AI draft, the system creates an `EDIT_USER` version.

- **Rule**: Manual edits cannot be overwritten by automated "Regenerations" unless the user explicitly archives the manual version.

3. Retention

- The system keeps the last 5 versions of any snapshot.
- Older versions are automatically purged unless "Pinned" by the user.

FILE: docs/06_AI/PROMPTS/SYSTEM/SYSTEM_CITATION_ENGINE.md

System Prompt: Citation Engine

Role and instructions for the citation anchoring component.

1. Role

You are the **Memoir.ai Citation Engine**. Your primary goal is to identify factual claims in a draft and anchor them to specific Evidence IDs from the provided dataset.

2. Core Instructions

- **Never Guess**: If a claim is made (e.g., "The cat was black") but the evidence does not state the color, do NOT add a citation.
- **Strict Logic**: A citation `[ID]` is only valid if the corresponding event directly supports the claim in the sentence.
- **Format**: Return the text with `[ID]` markers placed AFTER the specific claim being supported.

3. Evidence Context

The evidence is provided as a JSON array of `Events`.

- `EventID`: Use this for the bracketed citation.
- `Content`: The raw text to verify.
- `Timestamp`: Use for temporal verification.

4. Output Contract

Your output must be the original draft text with bracketed citations injected. If a sentence has NO supporting evidence, do NOT inject a citation.

FILE: docs/06_AI/PROMPTS/SYSTEM/SYSTEM_IMPORT_SUMMARIZER.md

System Prompt: Import Summarizer

Logic for generating "Highlights" during the data ingestion phase.

1. Role

You are the **Memoir.ai Evidence Profiler**. You process large volumes of imported data to provide a "Sense of Content" for the user's new vault.

2. Tasks

- **Summary**: Create a 2-sentence summary of the archived messages.
- **Top Contacts**: Identify the people the user communicated with the most.
- **Sentiment Shift**: Detect if the tone of messages changed significantly over time (e.g., "The archive starts energetic and becomes more professional toward the end").

3. Constraints

- **Privacy**: Do not store summaries outside the encrypted vault.
- **Redaction**: Replace any banking digits or passwords found in the data with `[REDACTED]`.
- **Neutrality**: Be descriptive and non-judgmental.

FILE: docs/06_AI/PROMPTS/SYSTEM/SYSTEM_SNAPSHOT_WRITER.md

System Prompt: Snapshot Writer

The core narrative soul of Memoir.ai.

1. Personality: The Nebula Voice

- **Tone**: Cinematic, reflective, precise, and warm.
- **Design Philosophy**: The voice should feel like a high-end documentary script—observational rather than conversational.

2. Drafting Rules

- **Evidence Anchoring**: Start every paragraph by grounding it in a time/place found in the data.

- **Lyrical Accuracy**: You may use evocative language ("The cold winter of 2019"), but ONLY if the data confirms the date and the presence of winter-related events.
- **Interpretations**: Labeled clearly. (e.g., "The data suggests a period of quiet reflection...").

3. Structuring

- Use headers to denote significant shifts in the timeline.
- Organize content chronologically unless the user requests a "Thematic" summary.

FILE: docs/06_AI/PROMPTS/TASK/PROMPT_CHAPTER_DRAFT.md

Task Prompt: Chapter Draft

Instructions for generating long-form memoir sections.

1. Objective

Generate a 500-word "Chapter" based on the provided 6-month time slice.

2. Variables

- `SUBSET_EVENTS`: The raw data.
- `CHAPTER_TITLE`: (e.g., "The Move to London").
- `USER_REFLECTIONS`: (Optional notes provided by the user).

3. Success Criteria

- [ ] At least 5 distinct citations to different sources.
- [ ] Clear beginning, middle, and end.
- [ ] Adherence to the "Nebula" cinematic voice.
- [ ] Inclusion of at least one media reference (e.g., "The photos from that park visit...").

FILE: docs/06_AI/PROMPTS/TASK/PROMPT_RELATIONSHIP_ARC.md

Task Prompt: Relationship Arc

Instructions for analyzing communication between two people.

1. Objective

Summarize the history and emotional trajectory of communication with `PERSON_ID`.

2. Key Questions

- When was the most intense period of communication?
- What were the recurring themes (work, family, shared interests)?
- How has the frequency of contact changed in the last 12 months?

3. Output Format

- **The Overview**: Brief summary of the connection.
- **Key Milestones**: List of 3-5 pivotal moments supported by citations.
- **The Trend**: Description of the current state of the relationship.

FILE: docs/06_AI/PROMPTS/TASK/PROMPT_TIMELINE_SUMMARY.md

Task Prompt: Timeline Summary

Instructions for generating condensed "Time Slice" highlights.

1. Objective

Summarize a specific day, week, or month into a dense, 100-word "Nebula Highlight".

2. Constraints

- **Format**: Single paragraph.
- **Citations**: REQUIRED for every claim.
- **Focus**: Emphasize unique events over repetitive daily routines.

3. Success Metric

The user should be able to scan their timeline and read these summaries to quickly remember the "vibe" of that period.

FILE: docs/06_AI/EVALUATION/EVAL_PLAN.md

AI Evaluation Plan (EVAL_PLAN)

Continuous verification of narrative quality and factual fidelity.

1. Methodology

Evaluation is performed locally during the build process and periodically on live user data (anonymized).

2. Golden Test Sets

Memoir.ai maintains 50+ "Golden Archives" (synthetic data with known ground truths).

- **Test 1**: Verify that a narrative about a 2018 trip correctly cites the 2018 messages.
- **Test 2**: Verify that the AI correctly identifies "Mom" even when she is referenced by multiple phone numbers.

3. Quantitative Targets

- **Claim Accuracy**: > 98% (Measured by Citation mismatch rate).
- **Hallucination Rate**: < 1% (Measured by internal NLI guards).
- **Latency**: Snapshot generation < 10 seconds on M1 hardware.

FILE: docs/06_AI/EVALUATION/HALLUCINATION_GUARDS.md

Hallucination Guards

Automated post-processing checks to prevent AI fabrications.

1. NLI Verification (Natural Language Inference)

For every generated sentence `S`:

1. Extract `Cited_Evidence_E`.
2. Run a local mini-model (Entailment check) to verify if `E` entails `S`.
3. **Action**: If `Contradicts`, mark the sentence as `UNVERIFIED_HALLUCINATION`.

2. Entity Matching Guard

- **Check**: Compare names and dates in the AI output against the raw metadata.
- **Action**: Hard fail if the AI invents a participant or date that does not exist in the provided `evidence_set`.

3. Tone Guard

- **Check**: Flag moralizing or overly assumptive language (e.g., "You were clearly angry").
- **Action**: Soft-rewrite recommendation to more neutral language (e.g., "The exchange was heated").

FILE: docs/06_AI/EVALUATION/QUALITY_RUBRICS.md

Quality Rubrics (AI)

Scoring criteria for narrative excellence.

1. Factual Fidelity (Weight: 50%)

- **5 - Exceptional**: Every claim perfectly supported by precise citations.
- **1 - Poor**: Multiple factual errors or dates shifted by > 24 hours.

2. Narrative Flow (Weight: 20%)

- **5 - Exceptional**: Smooth transitions, thematic coherence, cinematic "Nebula" tone.
- **1 - Poor**: Bulleted list style or repetitive sentence structure.

3. Citation Density (Weight: 30%)

- **5 - Exceptional**: ≥ 1 citation per sentence.
- **1 - Poor**: Large blocks of text with no verifiable supporting evidence.

4. Overall Grade

A snapshot is **"Verified"** only if its total score is ≥ 4.2/5.0.

FILE: docs/06_AI/SAFETY/SENSITIVE_CONTENT_POLICY.md

Sensitive Content Policy (AI)

Guardrails for handling delicate personal history elements.

1. Forbidden Inference Categories

The AI model is explicitly instructed (via System Prompt) to NEVER:

- **Diagnose**: Do not provide medical, psychological, or legal diagnoses based on chat history.
- **Speculate on Malice**: Do not attribute criminal intent or extreme negative motives without explicit corroborating text.
- **Financial Advice**: Do not provide investment or tax guidance based on bank/email history.

2. PII Redaction

- **Rule**: If a user generates a "Public Summary", the AI MUST redact full phone numbers, home addresses, and credit card digits using placeholder tags (e.g., `[REDACTED_PHONE]`).

3. High-Intensity Filtering

Optional "Sensitive Mode" in Settings:

- **Action**: The AI will skip events flagged as "Highly Emotional" or "Arguments" when generating summaries, focusing only on neutral/positive life milestones.

FILE: docs/06_AI/SAFETY/USER_CONTROL_GUARDRAILS.md

User Control Guardrails (AI)

Ensuring the user remains the ultimate editor of their history.

1. Affirmative Edits

The system MUST allow the user to manually override any AI-generated claim.

- **Action**: If a user edits a sentence, the associated `Citation` is marked as `USER_VERIFIED` and the AI is blocked from regenerating that specific block unless the user resets it.

2. Regeneration Constraints

Users can trigger "Regenerate" at three levels:

- **Sentence**: Only re-roll the selected claim.
- **Section**: Re-draft an entire chapter/month.
- **Global**: Start the entire snapshot from scratch with new parameters.

3. Opt-Out

Users can completely disable AI "Narrativization" in Settings, reverting the app to a "Strict Timeline Viewer" with no automated insights.

FILE: docs/07_SECURITY_PRIVACY/ACCESS_CONTROL.md

Access Control Policy

Ensuring that only authorized users can touch vault data.

1. Physical Access

Access control relies on the OS-level file permissions of the vault directory.

- **Requirement**: The application MUST set the vault directory permissions to `600` (User Read/Write only) on POSIX systems.

2. Logical Access

- **Passphrase Strength**: Minimum 12 characters, requiring a mix of complexity (Entropy ≥ 60 bits).
- **Auto-Lock**: Users can configure the app to lock the vault after 5, 15, or 60 minutes of inactivity.

3. Remote Access

- **Constraint**: Memoir.ai possesses NO mechanisms for remote access. No SSH, no cloud backdoors, no "Remote Desktop" support tools.
- **Exception**: User-initiated screen sharing during technical support (entirely outside app control).

FILE: docs/07_SECURITY_PRIVACY/AUDIT_LOGGING.md

Audit Logging Policy

Tracking internal actions for security and debugging without compromising privacy.

1. Scope of Audit

The system MUST log the following events to the local `audit.db`:

- **Auth Events**: Successful/Failed vault unlocks, passphrase changes.
- **Data Events**: Import job initiation, export generation, vault deletion.
- **System Events**: Key rotation, software update application.

2. Redaction Standards

To protect user privacy even from local logs:

- **NO Content**: Message payloads, participant names, or file paths are NEVER written to the audit log.
- **Hashes**: Use `SHA-256` hashes of `event_ids` if a specific record must be referenced for debugging.

3. Retention

- Audit logs are retained for **90 days** by default, after which they are automatically rotated and purged.

- Users can manually clear the audit log via Settings > Advanced.

FILE: docs/07_SECURITY_PRIVACY/COMPLIANCE_NOTES.md

Compliance Notes (GDPR/CCPA)

Mapping Memoir.ai’s local-first architecture to global privacy regulations.

1. Data Controller vs. Processor

- **User as Controller**: Since all data is stored on the user's personal hardware, the user is the sole controller of their data.
- **App as Tool**: Memoir.ai acts as a processing tool that provides functionality without seeing the data.

2. GDPR Rights Support

- **Right to Access**: Fulfilled via the "Export Data" feature (JSON/Markdown).
- **Right to Erasure (Right to be Forgotten)**: Fulfilled via the "Delete Vault" feature, which performs a secure local wipe.
- **Right to Portability**: Fulfilled via the non-proprietary CSV/JSON export schema.

3. Zero-Knowledge Compliance

As the vendor (Memoir.ai) never receives user data, we cannot fulfill standard "Search Warrants" or "Supoenas" for user content, as it is technically impossible for us to access.

FILE: docs/07_SECURITY_PRIVACY/DATA_RETENTION_DELETION.md

Data Retention & Deletion Policy

Defining the lifecycle of personal history within the vault.

1. Retention Period

- **Default**: Infinite. Once imported, data remains in the encrypted vault until the user chooses to delete it.
- **Auto-Cleanup**: Optional setting to automatically delete messages older than X years (Disabled by default).

2. Secure Deletion

- **Granular Deletion**: Users can delete individual events or conversation threads from the timeline.
- **Physcial Deletion**: When an event is deleted, its record is removed from the SQLite database, and its associated media is checked for orphan status before being unlinked from the file system.

3. Vault Wipe

A "Total Wipe" action:

1. Closes all active database connections.
2. Deletes the primary `.sqlite` vault file.
3. Deletes the entire `attachments/` subdirectory.
4. Purges all local caches and preference files.

FILE: docs/07_SECURITY_PRIVACY/ENCRYPTION_STRATEGY.md

Encryption Strategy

The cryptographic foundation of Memoir.ai.

1. Database-Level Encryption (At Rest)

- **Engine**: SQLCipher (commercially supported build).
- **Algorithm**: AES-256-GCM.
- **KDF**: PBKDF2 with 256,000 iterations (configurable) or Argon2id for higher security environments.

2. Key Derivation

The vault passphrase is never stored on disk.

1. User enters passphrase.
2. App derives a 256-bit key using the configured KDF and a local salt.
3. Key is used to `PRAGMA key` the SQLite session.

3. Media Encryption

- Attachments are stored as individual files in a hidden directory.
- **Encryption**: Files are individually encrypted with AES-256 before being written to disk. The decryption keys for media are stored WITHIN the encrypted SQLite database.

4. In-Memory Protection

- Sensistive key material is marked as `NON_SWAPPABLE` where the OS allows.
- Memory used for decryption buffers is zeroed out immediately after use.

FILE: docs/07_SECURITY_PRIVACY/EXPORT_GUARANTEES.md

Export Guarantees

Contractual commitments to data portability and openness.

1. No Data Lock-in

Memoir.ai guarantees that users can export 100% of their ingested and generated data at any time, even if their subscription has expired.

2. Readable Formats

All exports MUST include:

- **Events**: Pure JSON Schema.
- **Narratives**: Markdown (.md) or Plain Text (.txt).
- **Media**: Original file formats (JPG, PNG, MP4, MP3).

3. Offline Export

Exporting functionality is built into the local core and requires ZERO outbound network calls to complete.

FILE: docs/07_SECURITY_PRIVACY/INCIDENT_RESPONSE.md

Incident Response Plan (Local)

Procedures for technical failure and security breaches on the user's machine.

1. Vault Corruption

- **Detection**: The app fails to open the SQLCipher DB or integrity checks fail.
- **Action**:
    1. Prompt user to "Attempt Repair".
    2. Attempt to rebuild indices from raw event logs.
    3. If repair fails, guide user to restore from the most recent `.vault.backup`.

2. Suspected Local Breach (Malware/Unauthorized Access)

If a user suspects their local machine is compromised:

1. **Isolate**: Disconnect from the internet.
2. **Export**: Export critical narratives to a fresh, external hardware-encrypted drive.
3. **Wipe**: Perform a full "Vault Wipe" within the app to prevent further extraction.

3. Technical Support

- Memoir.ai technicians will NEVER ask for your passphrase.
- To help debug, users may manually export a "Sanitized Diagnostic Bundle" which includes only software versions and system error codes—NEVER private data.

FILE: docs/07_SECURITY_PRIVACY/PRIVACY_MODEL.md

Privacy Model

The core "Local-First" contract with the user.

1. Zero Cloud Dependency

- **Requirement**: The application must be fully functional for search, navigation, and local AI narrative generation without an active internet connection.
- **Verification**: The app uses on-device LLMs (e.g., local model wrappers) and local vector stores.

2. Telemetry & Analytics

- **Policy**: ZERO telemetry is sent by default.
- **Opt-In**: If the user opts-in to "Beta Feedback", only anonymized binary error codes and performance metrics are transmitted.
- **NO Content**: Message text, filenames, and participant names are NEVER transmitted, even in opt-in mode.

3. Data Integrity & Citation

- All AI narratives must be citation-backed to ensure that personal history is "Narrativized" but not "Fictionalized".
- If the AI cannot find enough evidence, the privacy policy mandates the system display an "Insufficient Evidence" warning rather than guessing.

FILE: docs/07_SECURITY_PRIVACY/THREAT_MODEL.md

Threat Model

Identifying and mitigating potential risks to the Memoir.ai vault.

1. Primary Threats & Mitigations

Threat	Description	Mitigation
Local Unauthorized Access	Someone opens the app on a shared machine.	Mandatory Vault Passphrase + Auto-Lock.
Physical Disk Theft	Vault folder is stolen from the computer.	SQLCipher AES-256-GCM encryption at rest.
Malformed Import File	Parser exploitation via a crafted ZIP/DB.	Sandbox parsing & structural validation before insert.
Hostile Malware	RAM scraper on the OS searches for keys.	Memory sanitization; zeroing out key material immediately.
Passphrase Loss	User forgets their key.	Local backup requirement; explicit "Unrecoverable" warnings.

2. Out of Scope

- Attackers with full kernel-level access to the machine.
- Nation-state level persistent hardware exploitation.
- OS-level vulnerabilities outside of application process control.

3. Vulnerability Reporting

Memoir.ai maintains a local-first philosophy, but security vulnerabilities in our code can be reported via `security@memoir.ai`.

FILE: docs/08_BILLING_AUTH/AUTH/RBAC_ENTITLEMENTS.md

RBAC & Entitlements

Defining the functional permissions based on subscription tiers.

1. Feature Entitlements

Feature	Free Tier	Pro Tier	Cinematic/Team
Max Sources	2	Unlimited	Unlimited
AI Snapshots	No	100 / Month	Unlimited
Advanced Search	Basic	Semantic	Semantic + Visual
Export Formats	JSON	All (+Markdown)	All + CSV
Cloud Backups	No	Optional Cloud Proxy	Shared Team Vault

2. Enforcement Logic

- **Local Cache**: Entitlements are cached in the encrypted vault metadata.
- **Validation**: Every `GEN_SNAPSHOT` or `IMPORT_JOB` request first checks the `subscription_status` against the local feature matrix.

3. Downgrade Behavior

If a user's Pro subscription expires:

- Their existing 100+ sources remain READABLE.
- They cannot add a NEW source until they are below the Free Tier limit or re-subscribe.
- AI narrativization functionality is locked.

FILE: docs/08_BILLING_AUTH/AUTH/SESSION_SECURITY.md

Session Security

Protecting the active vault state during runtime.

1. Vault Unlock 生命周期 (Lifecycle)

1. **Unlock**: Passphrase creates a temporary SQLCipher session key.
2. **In-Memory Storage**: The key is stored in a `Uint8Array` in the main process, never passed to the renderer.
3. **Heartbeat**: The renderer sends a "Session Heartbeat" every 30 seconds.
4. **Auto-Lock**: If no heartbeat is received or the inactivity timer expires, relevant memory is cleared and the DB connection is severed.

2. Process Isolation

- **Context Isolation**: Enabled in Electron.
- **Sandboxing**: Renderer processes have no direct access to Node.js or the file system.

- **IPC Validation**: All IPC messages from the UI must pass schema validation before the Main process executes them.

3. Lock State

When locked, the app presents a "Nebula" themed overlay. No raw data is visible in the UI background (blurred or un-rendered).

FILE: docs/08_BILLING_AUTH/AUTH/SUPABASE_AUTH_MODEL.md

Supabase Auth Model

Integration for non-vault metadata and billing synchronization.

1. Purpose

While the **Vault** is local-first, Memoir.ai uses Supabase for:

- User Account Management (Email verification).
- Subscription State synchronization (via Stripe hooks).
- Optional "Cloud Proxy" metadata (for users who choose to enable encrypted cloud backups).

2. Flow

1. **Signup/Login**: User authenticates with Supabase (JWT).
2. **ID Mapping**: The Supabase `user_id` is linked to the local `vault_id`.
3. **Sync**: On app launch, the app pulls the current `subscription_tier` and `usage_limits` from Supabase to the local entitlements cache.

3. Privacy Boundary

- **Raw content NEVER goes to Supabase**.
- Only high-level usage numbers (e.g., `num_events`, `num_snapshots`) are pushed to Supabase if the user has a paid account (required for billing verification).

FILE: docs/08_BILLING_AUTH/BILLING/INVOICING_REFUNDS_POLICY.md

Invoicing & Refunds Policy

Commercial terms for Memoir.ai premium services.

1. Invoicing

- **Delivery**: All invoices are generated via Stripe and emailed to the user's primary account email.
- **Local Access**: A PDF mirror of the last 12 invoices is cached locally in the `vault/billing/` directory for offline access.
- **Taxation**: Prices include relevant sales tax/VAT based on the user's billing address.

2. Subscription Billing

- **Cycle**: Monthly or Annual.
- **Auto-Renewal**: Enabled by default; can be disabled at any time via the In-App Billing Dashboard.

3. Refunds

- **Trial Period**: 14-day full refund policy for first-time annual subscribers if the product does not meet expectations.
- **Method**: Refunds are processed back to the original payment method through Stripe.
- **Policy**: No partial refunds for mid-cycle cancellations.

FILE: docs/08_BILLING_AUTH/BILLING/PAYWALL_RULES.md

Paywall Rules

Triggering points for monetization within the application.

1. Trigger: Source Limit

- **Condition**: User attempts to add a 3rd `DataSource` on the Free Tier.
- **Action**: Display "Nebula" themed paywall modal with a 30-day "Pro" trial pitch.

2. Trigger: AI Narratives

- **Condition**: User clicks "Generate Snapshot" on the Free Tier.
- **Action**: Full-screen landing page highlighting the "Cinematic Narratives" feature.

3. Trigger: Semantic Search

- **Condition**: User uses the ~ semantic prefix or advanced search filters.
- **Action**: Inline "Pro Feature" tag with a "Free Preview" (first 3 semantic searches are free).

4. Paywall Aesthetics

- **Design**: High-vibrancy Magenta gradients, smooth glassmorphism, and a "Cinematic" video background illustrating premium features.

FILE: docs/08_BILLING_AUTH/BILLING/STRIPE_INTEGRATION.md

Stripe Integration & Entitlements — Memoir.ai V1

1. Webhook Management

The system uses a dedicated Edge Function (`/webhooks/stripe`) to handle subscription events.

Event	Action	Target Table
`customer.subscription.created`	Create/update customer record; grant entitlements.	`app_public.customers`
`customer.subscription.updated`	Update plan and status; adjust usage limits.	`app_public.customers`
`customer.subscription.deleted`	Revert status to 'canceled'; downgrade entitlements.	`app_public.customers`

2. Entitlement Gating

Backend APIs must verify a user's plan and remaining usage before executing costly or restricted operations.

Key Feature Gates

- **AI Snapshot Count**: Monthly limit based on plan (e.g., Free Plan = 5 snapshots/month).
- **Library Creation**: Limit on the number of personal workspaces.
- **Export Access**: Pro-tier only or metered for free users.

3. Usage Tracking

Usage for AI services (Snapshots) and storage is tracked in real-time. Usage metrics are reconciled against the user's current plan entitlements during every protected request (e.g., `POST /libraries/:id/snapshots`).

FILE: docs/08_BILLING_AUTH/BILLING/STRIPE_PRODUCTS_PRICES.md

Stripe Products & Prices

Pricing configuration for the current release.

1. Product Matrix

Product ID	Name	Price (Monthly)	Price (Annual)
`prod_solo_v1`	**Solo (Free)**	$0.00	$0.00
`prod_pro_v1`	**Pro**	$14.00	$120.00
`prod_cin_v1`	**Cinematic**	$49.00	$450.00

2. Price IDs (Internal)

The app must map these IDs to the local `subscription_tier` enum within the `entitlements` system.

3. Discount Policies

- **Education/Student**: 50% discount available via manual verification.
- **Early Adopter**: Lifetime 25% discount for users who join during the Alpha phase.

FILE: docs/08_BILLING_AUTH/BILLING/SUBSCRIPTION_STATES.md

Subscription States

The state machine for user commercial status.

1. State Definitions

- **`ACTIVE`**: All premium features enabled.
- **`PAST_DUE`**: Payment failed; 7-day grace period with full access.

- **`CANCELED`**: User initiated cancellation; access remains until end of cycle.
- **`EXPIRED`**: Cycle ended; system reverts to Free Tier entitlements.
- **`TRIALING`**: Temporary Pro access (14 days); "Trial Ending" banners active.

2. Transition Rules

- **Free -> Pro**: Requires immediate Stripe checkout.
- **Pro -> Free (Downgrade)**: User data remains READABLE (see RBAC policy).
- **Pro -> Cinematic**: Pro-rata pricing applied in Stripe.

3. Offline Handling

If the user is offline for > 48 hours, the app uses the `subscription_status` cached during the last successful sync.

FILE: docs/08_BILLING_AUTH/BILLING/USAGE_METERING.md

Usage Metering

Tracking consumable limits for premium features.

1. Metered Units

- **`AI_TOKEN_COUNT`**: Total words generated by local AI models.
- **`SOURCE_COUNT`**: Number of active data sources in the vault.
- **`STORAGE_GB`**: Total size of the compressed vault + media.

2. Local Count vs. Cloud Sync

- **Local**: Usage is updated in real-time in the `vault_metadata` table.
- **Sync**: Every 24 hours (or on significant change), the high-level counts are pushed to the Supabase backend for billing verification.

3. Overage Logic

- **Hard Limit**: If `AI_TOKEN_COUNT` exceeds the monthly Pro limit (e.g., 50,000 words), generation is disabled until the next billing cycle.
- **Soft Limit**: Storage warnings appear at 80% usage.

FILE: docs/09_DEVOPS_RELEASE/CI_CD.md

CI/CD Pipeline Spec — Memoir.ai

Memoir.ai uses a robust CI/CD pipeline to ensure that every release of the desktop application is secure, stable, and high-performance.

1. Pipeline Overview

The pipeline is designed for an Electron-based cross-platform desktop app.

```
graph LR
    A[Code Push] --> B[Lint & Format]
    B --> C[Unit Tests]
    C --> D[Security Audit]
    D --> E[Build Electron App]
    E --> F[Verification Testing]
    F --> G[Release Artifacts]
```

2. Stages

Quality & Security

- **Linting**: Standard JS/React linting (ESLint, Prettier).
- **Unit Testing**: Jest/Vitest for React components and Node.js logic.
- **Dependency Checks**: Regular audits of `npm` packages for vulnerabilities.
- **Security Scans**: Verification that no secrets or API keys are leaked into the frontend code.

Automated Build

- **Electron Forge/Builder**: Used to package the application for macOS and Windows.
- **Code Signing**: Ensuring binaries are signed for trustworthy installation.

Verification

- **Smoke Tests**: Automated check that the app launches and the intro wizard initializes.
- **DB Check**: Verifying that the SQLCipher encryption initialization works on target OS.

3. Release Strategy

- **CI Environment**: GitHub Actions or similar local CI runner.
- **Artifacts**: ZIP/DMG/EXE files ready for manual or auto-update distribution.
- **Rollback**: The "Rollback Plan" involves reverting to the last tagged stable version on the CI runner.

FILE: docs/09_DEVOPS_RELEASE/DEPLOYMENT_RUNBOOK.md

Deployment Runbook (Desktop App)

Step-by-step procedures for building and distributing Memoir.ai.

1. Build Prerequisites

- Node.js v20+
- macOS: Xcode + Apple Developer Certificate (for signing).
- Windows: Windows SDK + EV Code Signing Cert.

2. Compilation Flow

1. **Dependency Check**: `npm audit` and license scan.
2. **Frontend Build**: `npm run build:ui` (React + Vite).
3. **Backend Bundle**: `npm run build:main` (Node + SQLCipher binaries).
4. **Electron Packager**: `electron-builder` constructs the `.app` / `.exe`.

3. Signing & Notarization

- **macOS**: `xcrun notarytool` submission to Apple.
- **Windows**: SignTool for EV signatures.

4. Distribution

- Update `latest.yml` / `latest-mac.yml` on the distribution server.
- Upload binaries to the secure download CDN.
- Trigger "Internal Alpha" auto-update pulse.

FILE: docs/09_DEVOPS_RELEASE/ENVIRONMENTS.md

Environments Specification

Configuration profiles for development, test, and production.

1. Local Development (`.env.development`)

- **Mode**: `DEBUG`
- **Vault Path**: `~/.memoir/debug_vault/`
- **AI Model**: External proxy (for speed) or high-latency local mode.
- **DevTools**: Enabled.

2. Beta / Test (`.env.test`)

- **Mode**: `TEST`

- **Vault Path**: Temporary sandbox folders.
- **Database**: Clean-on-start SQLCipher DB.
- **Diagnostics**: Verbose audit logging to `stdout`.

3. Production (`.env.production`)

- **Mode**: `PROD`
- **Vault Path**: User-selected directory.
- **AI Model**: Optimized for local hardware (Metal/CUDA/Vulkan).
- **Security**: Context isolation and stricter CSP enforced.

FILE: docs/09_DEVOPS_RELEASE/LOGGING_TRACING.md

Logging & Tracing

Local observability for the Memoir.ai backend.

1. Log Levels

- **`FATAL`**: Vault corruption or crash (Always logged).
- **`WARN`**: Import failures, failed logins.
- **`INFO`**: Job start/end, app lifecycle events.
- **`DEBUG`**: Detailed IPC message flow (Development only).

2. Storage

Logs are stored in the user's local application data directory (`userData/logs/`).

- **Filename**: `app.log` (Current) and `app.1.log` (Rotated).
- **Encryption**: Logs are NOT encrypted by default (must not contain private data).

3. Tracer

For complex AI jobs, the system uses an internal **Job Tracer** that tracks latency across the Pipeline (Extraction -> Inference -> Verification).

FILE: docs/09_DEVOPS_RELEASE/MONITORING_ALERTING.md

Monitoring & Alerting (Local)

Health checks for the individual user's vault.

1. Health Checks

The app monitors:

- **Disk Space**: Alert user if < 5GB remaining.
- **Database Latency**: Alert if query time > 500ms (suggests index rebuild).
- **Process Memory**: Auto-restart background workers if they exceed 2GB.

2. User Alerts

- **UI Toasts**: Low-criticality (e.g., "Import finished").
- **System Notifications**: High-criticality (e.g., "Vault Locked" or "Backup Failed").

3. Remote Analytics (Opt-In)

If enabled, the app sends high-level "Stability Hearts" (Heartbeats) to the central monitoring dashboard to track global crash rates without seeing user data.

FILE: docs/09_DEVOPS_RELEASE/RELEASE_GATES.md

Release Gates

Requirements for promoting a build to the production channel.

1. Quality Gates

- **Tests**: 100% pass rate in the `REGRESSION_CHECKLIST`.
- **Sentry**: zero "Unresolved" fatal issues in the current Beta release.
- **Performance**: P95 Search latency < 200ms on a 50k record test set.

2. Security Gates

- **Signing**: Valid Apple/Microsoft developer signatures.
- **Checksums**: Match between CI build and final distribution binaries.
- **Vulnerability Scan**: Zero high-criticality CVEs in the Node.js dependencies.

3. Human Gate

- Product Manager approval on "Final Walkthrough" for new UI features.

FILE: docs/09_DEVOPS_RELEASE/ROLLBACK_PLAN.md

Rollback Plan

Procedures for recovering from a faulty release.

1. Rapid Rollback

If a fatal bug is detected after V[X] deployment:

1. Pause the auto-update server immediately.
2. Downgrade the "Latest" version pointer in the CDN to V[X-1].
3. Trigger an auto-update "Downgrade" (if supported) or notify users to download the prior version.

2. Database Compatibility

- **Rule**: If V[X] introduced a schema change, V[X-1] MUST be able to read the modified DB (Forward Compatibility) or the rollback must include a data migration script.
- **Recommendation**: Always perform a `.vault.backup` before applying major version updates.

3. Communication

- Post a "Known Issue" banner on the support site with a 1-click downgrade link.

FILE: docs/09_DEVOPS_RELEASE/SECRETS_MANAGEMENT.md

Secrets Management

Handling the high-value keys that secure Memoir.ai.

1. User Secrets (Local)

- **Vault Passphrase**: Never stored.
- **Master Key**: Generated via Argond2id/PBKDF2; kept in sanitized memory.

2. Corporate Secrets (Build Time)

- **Apple Certs**: Stored as protected GitHub Actions Secrets.
- **Stripe API Keys**: Injected via environment variables during CI build; NEVER hardcoded.
- **Supabase Keys**: Public anon key used locally; Secret Service key used only for backend webhooks.

3. Key Rotation

- **Policy**: The application rotated its internal session tokens every 24 hours of active use.
- **Recovery**: Users are guided to store an "Emergency Recovery Key" (offline) to reset their vault if the primary passphrase is lost.

FILE: docs/11_USER_DOCS/EXPORT_DELETE_GUIDE.md

Export & Deletion Guide

Total control over your digital legacy.

1. Exporting Your Data

- **The Master ZIP**: Navigate to Settings -> Privacy -> Export Vault.
- **The Format**: Your data is exported as human-readable JSON files along with all original media.
- **Integrity**: The export includes a cryptographically signed manifest to verify its completeness.

2. Deleting Your Vault

- **Action**: Settings -> Danger Zone -> Delete Vault.
- **Warning**: This action is IRREVERSIBLE. It performs a multi-pass overwrite of the local files.
- **Cloud Sync**: Deleting your vault also purges all linked metadata from the Supabase backend.

3. Portability Guarantee

We promise that your data will always be exportable in an open format. You will never be locked into Memoir.ai.

FILE: docs/11_USER_DOCS/GETTING_STARTED.md

Getting Started Guide

Welcome to your private time machine.

1. Installation

1. Download the `.dmg` (macOS) or `.exe` (Windows).
2. Move to your Applications folder.
3. Launch Memoir.ai.

2. Setting Up Your Vault

- **Action**: Click "Create New Vault."
- **Important**: Choose a strong passphrase. There is NO "Forgot Password" for your local data. If you lose this passphrase, your data is gone.

3. Your First Import

1. Navigate to the **Imports** tab.
2. Select "iMessage Backup."
3. Follow the onscreen instructions to locate your database.
4. Wait for the "Import Complete" notification.

4. Explore

Double-click any event on the Timeline to see the full context and associated media.

FILE: docs/11_USER_DOCS/IMPORT_GUIDE.md

Data Import Guide

How to bring your history into Memoir.ai.

Supported Sources

iMessage (macOS)

- **Path**: `~/Library/Messages/chat.db`
- **Requirement**: You must grant "Full Disk Access" to Memoir.ai in System Settings for this import to work.

WhatsApp

- **Method**: Go to WhatsApp -> Settings -> Chats -> Export Chat.
- **Action**: Import the resulting `.zip` file into Memoir.ai.

Raw JSON

- Follow the structure defined in CANONICAL_DATA_MODEL.md.

Troubleshooting

- **Missing Media**: Ensure photos were not offloaded to iCloud "Optimize Storage" at the time of import.
- **Slow Speed**: Large imports can take 10+ minutes. Close other heavy apps to speed up the process.

FILE: docs/11_USER_DOCS/SEARCH_GUIDE.md

Search Guide

How to find anything in your digital past.

1. Basic Search

Simply type in the top bar. We handle:

- **Names**: "Allison" or "Mom"
- **Locations**: "Paris" or "Home"
- **Keywords**: "Contract" or "Birthday"

2. Semantic Search (The Magic Search)

Search by feeling. Prefix with ~:

- `~the first time we met`
- `~when I was feeling stressed about work`
- `~sunset at the beach`

3. Filters

Refine your results by:

- **Date**: Specific months or year ranges.
- **Source**: Only iMessage, only WhatsApp, etc.
- **Media**: Only show results with attachments.

FILE: docs/11_USER_DOCS/SNAPSHOTS_GUIDE.md

Snapshots Guide

Turning data into cinema.

1. What is a Snapshot?

A Snapshot is an AI-generated narrative that summarizes a period of your life or a specific relationship.

2. Generating Your First Snapshot

1. Select a range on the Timeline.
2. Click "Generate Narrative."
3. Choose a **Tone** (Lyrical, Reflective, or Clinical).

3. Fact Checking

Every sentence in a snapshot is underlined. Click the underline to open the **Evidence Drawer**, which shows the exact raw messages used to generate that claim.

4. Editing

You can manually edit any sentence. The AI will respect your edits and won't overwrite them in future regenerations.

FILE: docs/11_USER_DOCS/TIMELINE_GUIDE.md

Timeline Guide — Memoir.ai

The **Unified Timeline** is the heart of Memoir.ai. It presents every memory, message, and event in a single, cinematic chronological flow.

1. Navigating the Feed

- **Infinite Scroll**: Your entire history is loaded dynamically as you scroll.
- **Date Jump**: Use the sidebar to jump to specific years or months.
- **Search Bar**: Located at the top, allowing for instant filtering.

2. Event Cards & Metadata

Each entry in the timeline is an "Event Card."

- **Identity**: See exactly which source the event came from (e.g., "WhatsApp", "Email").
- **Metadata Panel**: Click an event to open the sidebar, revealing raw JSON data, participants, and emotional sentiment analysis.
- **Citations**: Use the "Copy Citation" feature to link this specific memory in your narratives.

3. Conversation Reconstruction

For messaging apps, Memoir.ai stiches threads together.

- **Thread View**: Clicking a message opens the **Conversation Viewer**, showing the full back-and-forth reconstruction across platforms if applicable.
- **Attachments**: Inline previews for photos and voice notes are displayed directly in the timeline.

4. Filters & Facets

Use the Filter Panel to focus your view:

- **By Source**: View only iMessages or only emails.
- **By Emotion**: (AI-powered) Show only memories with "Joy" or "Analytical" sentiment.
- **By Person**: Filter the timeline to show interactions with a specific contact.

FILE: docs/11_USER_DOCS/TIMELINE_SEARCH_GUIDE.md

Timeline & Search Guide

Navigating your history with the Private Time Machine.

1. The Timeline

- **The Scrub Bar**: Use the vertical bar on the right to jump between years instantly.
- **The Unified Feed**: Photos and messages are merged into a single chronological stream.
- **Focus Mode**: Click the "Eye" icon on a participant to see only messages from them.

2. Advanced Search

- **Keyword Search**: Type any name or place.
- **Semantic Search**: Prefix with ~ for vibey matching (e.g., `~the day we moved`).
- **Filters**: Use the "Filter" button to restrict results by platform (iMessage vs WhatsApp) or date range.

3. Bookmarks

Click the "Star" icon on any event to save it to your "Core Memories" collection.

FILE: docs/11_USER_DOCS/TROUBLESHOOTING.md

User Troubleshooting FAQ

Common issues and solutions.

Vault Access

- **Q: "Invalid Passphrase" error but I'm sure it's right.**
- **A**: Check caps lock. Ensure you haven't moved the `.vault` file to a different directory.

Performance

- **Q: Moving through the timeline feels sluggish.**
- **A**: The app may be re-indexing your data. Check the Status Bar for background job activity.

AI Snapshots

- **Q: The AI is making things up.**
- **A**: Click the "Correction" icon on the snapshot and highlight the error. This helps tune the model for your specific writing style.

FILE: docs/12_PROMPT_LIBRARY_ASSETS/BUILD_AGENT_PROMPTS/BUILD_AGENT_PROMPTS.md

Build Agent Prompts

Instructions for specialized coding assistants working on Memoir.ai.

1. Documentation Agent

- **Instruction**: "Maintain the deep manifest structure. Every file must have a purpose section. Use Mermaid for flowcharts."

2. SQLCipher Migration Agent

- **Instruction**: "Always check `PRAGMA cipher_version` before executing migrations. Ensure all table names follow the `snake_case` plural convention."

3. UI Refactor Agent

- **Instruction**: "Strictly adhere to the 'Nebula' color palette (Violet/Cyan/Magenta). Do not use Tailwind default colors."

FILE: docs/12_PROMPT_LIBRARY_ASSETS/BUILD_AGENT_PROMPTS/ITERATIVE_BUILD_STEPS.md

Iterative Build Prompts — Memoir.ai V1

These prompts are designed to be executed sequentially by an agentic builder to construct Memoir.ai V1 from the ground up, following the Master Specification and manifest-defined modular docs.

Phase 1: Foundational Setup & Schema

Prompt 1.1: Project Initialization

> "Initialize a new Electron + React project with basic routing (`/auth`, `/libraries`, `/settings`). Integrate the Supabase client library and configure environment variables for Backend URL and Anon Key. Create a placeholder login screen."

Prompt 1.2: Database Foundation

> "Execute the canonical SQL schema found in `docs/04_DATA/CANONICAL_DATA_MODEL.md`. Crucially, enable Row Level

Security (RLS) on all tables and implement the policy to restrict operations to the owning `user_id` or `library_id`."

Phase 2: Auth & Workspace

Prompt 2.1: Authentication Flow

"Implement the full Sign Up and Login flow using Supabase Auth. Upon login, redirect to `/libraries`. Ensure all data access uses the authenticated user's token, confirming RLS is active."

Prompt 2.2: Library Management

"Build the Library selection/creation dashboard at `/libraries`. Implement logic for: 1. Creating a Library. 2. Selecting a Library (redirect to `/libraries/:id`). 3. Deleting a Library (with irreversible warning modal)."

Phase 3: Ingestion Pipeline

Prompt 3.1: Import Wizard

"Create a wizard-guided import modal in the Library workspace. It must: 1. Allow file selection/upload (CSV/JSON) to `memoir-ai-exports` bucket. 2. Initiate a job via `POST /libraries/:id/imports` with the signed storage URL."

Prompt 3.2: Job Monitoring

"Build the Job Monitoring component at `/libraries/:id/jobs`. Display a list of `import_jobs` with a `ProgressBar` linked to `progress_percentage`. Implement real-time updates via Supabase."

Phase 4: Timeline & AI Snapshots

Prompt 4.1: Unified Timeline & Search

"Develop the Timeline view at `/libraries/:id`. Fetch `memories` chronologically, grouped by date. Implement filters for Date range, Source, and Participants. Add a global Full-Text Search bar."

Prompt 4.2: AI Snapshot Generation

"Implement the AI Snapshot flow: 1. Select range/memories on timeline. 2. Initiate `POST /libraries/:id/snapshots`. 3. Display narrative with `CitationComponent` pills linked to source records."

Phase 5: Billing & Polish

Prompt 5.1: Stripe Integration

"Integrate Stripe Checkout and implement the `/webhooks/stripe` handler to update `app_public.customers`. Implement backend entitlement checks to block AI generation if usage limits are exceeded."

Prompt 5.2: Data Control & QA

> "Implement Library-level settings for Data Export and Permanent Deletion. Perform a final verification against the `docs/10_QA/TEST_MATRIX.md` to ensure all V1 constraints are met."

FILE: docs/12_PROMPT_LIBRARY_ASSETS/BUILD_AGENT_PROMPTS/MASTER_BUILD_PROMPT.md

Master Build Prompt

The foundational context for any AI agent interacting with the Memoir.ai repository.

1. Project Context

Memoir.ai is a local-first, privacy-focused personal historian.

- **Privacy Policy**: Never propose code that exfiltrates user data.
- **Aesthetic**: All UI must adhere to the "Nebula" theme.
- **Integrity**: Every AI output must be cited to raw evidence.

2. Technical Stack

- Electron + React + Node.js.
- SQLCipher (AES-256) for local storage.
- Supabase for non-vault sync.

3. Workflow Rules

- Always update relevant MANIFEST.md files when adding docs.
- Use Mermaid diagrams for complex logic flows.
- Maintain the absolute isolation between the Renderer and the File System.

FILE: docs/12_PROMPT_LIBRARY_ASSETS/BUILD_AGENT_PROMPTS/PROMPT_BACKEND_ENGINEER.md

Backend Engineer Build Prompt

Role-specific instructions for Node.js and IPC development.

1. Role Focus

You are an expert Backend Engineer optimizing the Ingestion and AI pipelines.

2. Core Responsibilities

- **Concurrency**: Use child processes for heavy parsing.
- **Security**: Validate all IPC messages at the bridge.
- **Performance**: Optimize SQL query plans for large datasets.

3. Style Guide

- Use TypeScript for all backend logic.
- Implement robust error handling with the project's ERROR_TAXONOMY.md.

FILE: docs/12_PROMPT_LIBRARY_ASSETS/BUILD_AGENT_PROMPTS/PROMPT_DB_ENGINEER_SUPABASE.md

Agent Persona: Supabase DB Engineer

You are a **Database Architect & Security Specialist** specializing in PostgreSQL and the Supabase ecosystem. You protect the "Digital Memory" of our users.

1. Responsibilities

- Design scalable, normalized PostgreSQL schemas.
- Implement **Row Level Security (RLS)** policies for every table (The multi-tenant firewall).
- Write efficient SQL functions (plpgsql) and triggers.
- Manage database migrations and version control.

2. Technical Standards

- **Schema Safety**: Use UUIDs for Primary Keys.
- **Indexing**: Proactively index foreign keys and columns used in filtering.
- **RLS**: Defaults to `DENY`. Every table MUST have a policy allowing only the `auth.uid()`.
- **Integrity**: Enforce strong constraints (NOT NULL, UNIQUE, CHECK).

3. Privacy Focus

- **Encryption**: Coordinate with the Security Auditor for PGP encryption of sensitive memory fields at the DB level.
- **Audit Logs**: Maintain triggers for tracking changes to critical user settings.

- **Least Privilege**: Grant only necessary permissions to database roles (`anon`, `authenticated`, `service_role`).

4. SQL Patterns

- Use snake_case for table and column names.
- Prefer `auth.uid()` over passing user IDs in query parameters where possible.
- Avoid wide tables; normalize text-heavy memories into optimized chunks.

5. Collaboration

- Provide the **Backend Engineer** with optimized views and functions.
- Ensure the **Security Auditor** can easily trace data access patterns through your schema design.

FILE: docs/12_PROMPT_LIBRARY_ASSETS/BUILD_AGENT_PROMPTS/PROMPT_FRONTEND_ENGINEER.md

Agent Persona: Frontend Engineer

You are a **Senior Frontend Engineer & UI Specialist**. You craft the "Telescope"—the window through which users see their past.

1. Responsibilities

- Implement high-fidelity interfaces using **React** and **Next.js**.
- Enforce the **Cosmic Glass** design system (`theme.css`).
- Build responsive, fluid layouts that adapt to any screen.
- Orchestrate smooth, cinematic animations using CSS or Framer Motion.

2. Technical Standards

- **Component-Driven**: Build modular, reusable components with clear prop types.
- **Theme-Strict**: Use *only* variables from `theme.css`. Do not hardcode hex values.
- **Fluidity**: Implement typography and spacing using the `clamp()` tokens.
- **Performance**: Optimize for Core Web Vitals (LCP/CLS) even with heavy glass blurring.

3. Cinematic Aesthetic

- **Glassmorphism**: Correctly layer panels using `--panel`, `--panel-2`, and `--panel-3`.
- **Gradients**: Use `--grad-accent` for brand moments and `--grad-sheen` for hover states.
- **The Void**: Maintain the deep indigo background and noise texture to ground the UI.

4. Collaboration

- Consume APIs provided by the **Backend Engineer** with robust error handling and loading states.
- Follow the accessibility guidelines outlined in `COLOR_SYSTEM.md`.
- Coordinate with the **Master Agent** to ensure the "Feel" of the app is consistent.

5. Code Quality

- Clean, semantic HTML.
- Accessible ARIA labels for complex glass components.
- Zero-jump layouts using skeleton loaders.

FILE: docs/12_PROMPT_LIBRARY_ASSETS/BUILD_AGENT_PROMPTS/PROMPT_QA_ENGINEER.md

Agent Persona: QA Engineer

You are a **Precision Quality Engineer**. Your mission is to ensure Memoir.ai is bug-free, fast, and reliable. You are the final guardian before release.

1. Responsibilities

- Write comprehensive unit and integration tests.
- Identify edge cases in complex memory search and AI generation flows.
- Automate browser testing using tools like Playwright or Vitest.
- Profile performance for slow API responses or UI jank.

2. Testing Philosophy

- **Contract Testing**: Verify that Backend and Frontend speak the same JSON.
- **State Testing**: Ensure the UI handles "Empty", "Loading", "Error", and "Massive Data" states gracefully.
- **Regression**: Every bug found must have a corresponding test case to prevent recurrence.

3. Privacy Validation

- **Leak Detection**: Verify that data from User A never appears in User B's session.
- **Redaction Check**: Ensure PII is properly filtered in UI displays and console logs.
- **RLS Audit**: Work with the DB Engineer to verify that direct DB access respects security policies.

4. Operational Standards

- Report bugs with clear reproduction steps: **Symptom**, **Reproduction**, **Expected**, **Actual**.

- Validate responsive design across standard breakpoints (Mobile, Tablet, Desktop).
- Verify that animations in "Cosmic Glass" don't drop frames on low-end hardware.

5. Collaboration

- Challenge the **Backend Engineer** on error handling edge cases.
- Provide the **Frontend Engineer** with evidence of UI regressions.
- Assist the **Master Agent** in deciding if a build is "Ready for Orbit."

FILE: docs/12_PROMPT_LIBRARY_ASSETS/BUILD_AGENT_PROMPTS/PROMPT_SECURITY_PRIVACY_AUDITOR.md

Agent Persona: Security & Privacy Auditor

You are the **Lead Security Architect & Privacy Guardian**. You ensure that Memoir.ai remains a safe sanctuary for the user's most personal memories.

1. Responsibilities

- Audit every piece of code for potential data leaks or vulnerabilities.
- Enforce strict PII (Personally Identifiable Information) handling rules.
- Review Row Level Security (RLS) policies for complete isolation.
- Audit third-party dependencies for security flaws.

2. Security Pillars

- **Encryption**: Verify end-to-end encryption paths for sensitive data.
- **Least Privilege**: Ensure no agent or user has more access than required.
- **Observability**: Monitor for suspicious data access patterns (without compromising user content privacy).

3. Privacy Standards

- **Redaction**: Enforce the rule that No Unredacted PII (Emails, Phones, SSNs) should ever hit the logs.
- **Verification**: Data must only be accessible to the owner. Cross-user leakage is a Category Zero failure.
- **Anonymization**: Ensure AI models are fed only the data necessary for the request, with sensitive fields masked where possible.

4. Operational Focus

- Perform regular "Threat Modeling" on new features.
- Review and approve the **DB Engineer's** RLS policies.
- Sanitize all user-input paths to prevent SQL Injection or XSS in the "Cosmic" UI.

5. Collaboration

- Act as the "Hard Reset" on any feature that compromises privacy for convenience.
- Work with the **Backend Engineer** to implement secure auth flows.
- Advise the **Master Agent** on regulatory compliance (GDPR/CCPA/SOC2).

FILE: docs/12_PROMPT_LIBRARY_ASSETS/IMAGE_PROMPTS/DASHBOARD.md

Dashboard & Screen Prompts

1. Dashboard

- **Prompt**: "Modern software dashboard for digital history, dark mode amethyst theme, high-quality data visualization widgets, glassmorphism cards, blurred purple backgrounds, 8k UI design."

2. Empty States

- **Prompt**: "Set of 4 minimal icons for empty states, cosmic theme, glowing outlines, soft translucency, high contrast on black background."

3. Hero Image

- **Prompt**: "Cinematic wide shot of a person observing a glowing glass time machine, floating in a star field, deep violet and cyan lighting, hyper-realistic, 8k."

FILE: docs/12_PROMPT_LIBRARY_ASSETS/IMAGE_PROMPTS/EMPTY_STATES.md

Image Prompt: Empty States "Quiet Expectations"

Vision: A calm, inviting visualization for views with no content yet.

Prompt Details

> **Prompt**: A minimalist, high-end still life of a single crystalline floating leaf or star-shard resting on a pane of dark frosted glass. Extremely shallow depth of field, cinematic soft focus. One primary light source from above casting a soft cyan glow. Deep obsidian background with a subtle noise texture. Peaceful, sanctuary-like, ultra-modern. 8k, Octane render, photorealistic.

Technical Requirements

- **Subject**: Single minimalist object (leaf, shard, drop of light).
- **Lighting**: Single-point soft lighting, high contrast.
- **Vibe**: Calm, non-intrusive.

Usage

Use primarily for large empty state containers in the feed or catalog views.

FILE: docs/12_PROMPT_LIBRARY_ASSETS/IMAGE_PROMPTS/HERO_IMAGE.md

Image Prompt: Hero Image "The Telescope of Time"

Vision: The primary marketing image for Memoir.ai, capturing the "Cosmic Glass" essence.

Prompt Details

> **Prompt**: An intricate, futuristic telescope made of translucent violet glass and liquid silver. The lens is glowing with a swirling nebular energy in shades of cyan, magenta, and white. Floating in a vast, dark, cosmic void. Dust motes of light scattered in the background. Cinematic lighting, epic scale, hyper-detailed, 8k, ray-traced, masterpiece quality.

Technical Requirements

- **Primary Focus**: The "Telescope" (Brand metaphor).
- **Energy**: Swirling nebula effects.
- **Mood**: Epic, awe-inspiring, high-tech but magical.

Usage

Landing page hero sections, app store banners, or "About" pages.

FILE: docs/12_PROMPT_LIBRARY_ASSETS/IMAGE_PROMPTS/IMAGE_PROMPTS.md

Image Generation Prompts

Prompts for the marketing and UI assets (Nebula Theme).

1. "The Nebula Gateway"

- **Prompt**: "A cinematic hyper-realistic close-up of a glass sphere containing swirling violet and cyan nebulae, obsidian background, soft magenta lighting, 8k resolution, glassmorphism."

2. "Digital Archeology"

- **Prompt**: "Abstract representation of data streams flowing into a sleek, metallic vault, cosmic aesthetic, high contrast, minimalist UI style."

FILE: docs/12_PROMPT_LIBRARY_ASSETS/IMAGE_PROMPTS/LOADING_SCREEN.md

Image Prompt: Loading Screen "Nebula Spin"

Vision: A visualization of data being structured and loaded.

Prompt Details

> **Prompt**: A swirling vortex of bioluminescent star-dust in a perfect circle. Concentric rings of violet and cyan light particles suspended in a dark, hazy void. The center of the circle is deep obsidian. Macro photography style, soft glowing trails, cinematic motion-blur feel. Elegant, clean, futuristic. 8k resolution, minimalist.

Technical Requirements

- **Shape**: Circular (for loaders).
- **Motion**: Implied via particle trails.
- **Color**: Seamless blend between Cyan and Violet.

Usage

Splash screens, background for loading overlays.

FILE: docs/12_PROMPT_LIBRARY_ASSETS/IMAGE_PROMPTS/LOGIN_SCREEN.md

Image Prompt: Login Screen "The Key to the Vault"

Vision: A visualization of security, privacy, and the act of unlocking memories.

Prompt Details

> **Prompt**: A macro close-up of a human fingerprint etched in glowing violet neon lines onto a dark obsidian surface. The fingerprint is surrounded by a protective aura of translucent glass shards. Soft cyan backlighting creating a rim-light effect. Cinematic, mysterious yet secure. 8k, photorealistic, premium tech aesthetic.

Technical Requirements

- **Subject**: Fingerprint or key metaphor.
- **Materials**: Obsidian and glowing glass.
- **Mood**: Secure, private, high-fidelity.

Usage

Authentication pages, security settings headers.

FILE: docs/12_PROMPT_LIBRARY_ASSETS/UI_COPY_PROMPTS/ERROR_MESSAGES.md

Prompt: Error Message Generation (The Cosmic Guide)

You are tasked with generating error messages for Memoir.ai. Our goal is to handle failure with empathy, precision, and a "Cosmic Guide" voice.

1. Principles

1. **Empathy First**: Acknowledge that a failure is a disruption to the user's journey.
2. **Precision**: State exactly what went wrong without using cryptic system codes in the primary message.
3. **Actionable Recovery**: Always provide a clear "Next Step" or alternative path.
4. **No Blame**: It’s the system’s job to be resilient; never imply the user did something "wrong."

2. Cosmic Aesthetics

Use subtle cosmic metaphors where appropriate (e.g., "Signal loss," "Misalignment," "Deep space timeout") but never at the expense of clarity.

3. The 3-Part Structure

Every error message must follow this format:

1. **The Headline**: What happened? (Short, bold)
2. **The Description**: Why did it happen? (Simple sentence)
3. **The Action**: What do I do now? (Button text or link)

4. Examples

Scenario: Network Timeout during Import

- **Headline**: **Connection Lost in the Void**
- **Description**: We lost touch with the server while importing your memories. Don't worry, your progress is saved.
- **Action**: [Retry Sync]

Scenario: Invalid File Type

- **Headline**: **Unknown Format Detected**
- **Description**: This telescope can only read .ZIP or .JSON archives for now.
- **Action**: [View Supported Formats]

Scenario: Database (RLS) Permission Error

- **Headline**: **Access Denied**
- **Description**: This memory is private and can only be accessed by its owner.
- **Action**: [Return to Dashboard]

5. Instructions for the Agent

When generating an error message:

1. Identify the technical failure.
2. Translate it into the "Cosmic Guide" voice.
3. Ensure the action is clearly linked to a frontend component or route.

FILE: docs/12_PROMPT_LIBRARY_ASSETS/UI_COPY_PROMPTS/MICROCOPY_SYSTEM.md

System: Microcopy Generation (Memoir.ai)

This system governs the generation of labels, tooltips, buttons, and "bits" of text throughout the Memoir.ai "Cosmic Glass" interface.

1. Voice Checklist

- [ ] Is it minimal? (Fewer words = more space for memories).
- [ ] Is it sentence case? (Only capitalize the first word and proper nouns).
- [ ] Is it active? (Use dynamic verbs like "Explore", "Build", "Secure").
- [ ] Is it grounded? (Avoid tech-jargon).

2. Component Logic

Buttons (Primary Actions)

- Use **Verb + Noun** patterns.
- *Examples*: "Import memories", "Analyze thread", "Sync vault".
- Avoid generic "Submit" or "OK". Use "Confirm" or "Done".

Empty States (The Void)

- When a view is empty, use the copy to invite exploration.
- *Example*: "The cosmos is quiet. Start your first import to populate your timeline."

Tooltips (Guidance)

- Keep under 10 words.
- Focus on the "Why" or "How".
- *Example*: "Analyze this conversation for recurring themes."

3. Semantic Color Mapping

Match the copy to the design tokens:

- **Violet (Primary)**: Key navigational steps.
- **Cyan (Info)**: Meta-data explanations.
- **Magenta (Spark)**: AI-generated insights or creative suggestions.
- **Danger (Red)**: Destructive actions (e.g., "Remove permanently").

4. Instructions for the Agent

When drafting microcopy:

1. Reference the specific component (Button, Label, Tooltip).
2. Apply the "Sentence Case" rule.
3. Ensure the verb choice reflects the "Cosmic Guide" persona—authoritative but gentle.

FILE: docs/12_PROMPT_LIBRARY_ASSETS/UI_COPY_PROMPTS/ONBOARDING_COPY.md

Prompt: Onboarding Copy (The Cosmic Guide)

Onboarding is the user's first flight with Memoir.ai. Your goal is to guide them from "What is this?" to "This is my life’s archive" with 100% confidence.

1. The Narrative Flow

1. **Welcome**: Invitation to the sanctuary.
2. **Privacy Pledge**: The "Contract of Trust" (Your data is yours).
3. **Connection**: The first data import (Telegram, iMessage, etc.).
4. **Analysis**: Demonstrating the AI's power (The "Aha!" moment).
5. **Orbit**: Entering the main dashboard.

2. Voice Tone: "The Mentor"

During onboarding, the "Cosmic Guide" becomes a mentor. Use warmer, more encouraging language without becoming verbose.

3. Key Copy Blocks

The Privacy Pledge (Critical)

> "In the void of space, your data is your only anchor. At Memoir.ai, we never sell your memories. Everything is encrypted, private, and owned by you. Forever."

The "Aha!" Moment (Analysis)

> "Our AI is now scanning the nebulas of your conversations to find the stars that matter most—recurring jokes, deep connections, and forgotten milestones."

4. Onboarding Checklist

- Use **"We"** sparingly (to represent the guide).
- Use **"You"** and **"Your"** to put the user in control.
- Highlight the **"Value"** before the **"Effort"** (e.g., "See your year in review" before "Upload your logs").

5. Instructions for the Agent

When drafting onboarding steps:

1. Focus on the emotional outcome (Nostalgia, Clarity, Organization).
2. Reiterate privacy at every friction point (e.g., before an upload).
3. Keep the "Cosmic" theme alive but prioritize ease of use.

FILE: docs/12_PROMPT_LIBRARY_ASSETS/UI_COPY_PROMPTS/UI_COPY_PROMPTS.md

UI Copy Prompts

Guidelines for the AI when generating interface text.

1. Empty State Pitch

- **Goal**: Encourage the user to import their first source.
- **Keywords**: Sanctuary, History, Begin, Discovery.
- **Example**: "Your sanctuary is quiet. Begin by adding a data source to see your history bloom."

2. Onboarding Success

- **Goal**: Congratulate the user on securing their data.
- **Keywords**: Fortress, Sealed, Secure, Yours.
- **Example**: "Your vault is sealed. Your history is now yours alone."

FILE: scripts/db/apply_migrations_local.sh

```bash
#!/bin/bash
# apply_migrations_local.sh
# Applies pending migration files to the local SQLite/SQLCipher DB.

DB_PATH="./vault.db"
MIGRATIONS_DIR="./docs/04_DATA/SUPABASE/MIGRATIONS"

echo "Applying migrations to $DB_PATH..."
# Placeholder for migration logic
echo "Success: Mock migrations applied."
```

FILE: scripts/db/generate_migrations.sh

```
#!/bin/bash
# generate_migrations.sh
# Creates a new numbered migration file.

echo "Enter migration description:"
read DESC
CLEAN_DESC=$(echo "$DESC" | tr ' ' '_')
VERSION=$(date +%Y%m%d%H%M%S)

TOUCH_FILE="docs/04_DATA/SUPABASE/MIGRATIONS/${VERSION}_$
{CLEAN_DESC}.sql"
touch "$TOUCH_FILE"
echo "Created $TOUCH_FILE"
```

FILE: scripts/db/rls_lint_check.py

```
# rls_lint_check.py
# Validates that all .sql files in RLS_POLICIES contain an ENABLE ROW
LEVEL SECURITY statement.

import os

RLS_DIR = "docs/04_DATA/SUPABASE/RLS_POLICIES"

def check_rls():
    for filename in os.listdir(RLS_DIR):
        if filename.endswith(".sql"):
            with open(os.path.join(RLS_DIR, filename), 'r',
encoding='utf-8') as f:
                content = f.read()
                if "ENABLE ROW LEVEL SECURITY" not in content.upper():
                    print(f"FAIL: {filename} is missing RLS
enablement.")
                else:
                    print(f"PASS: {filename}")

if __name__ == "__main__":
    check_rls()
```

FILE: scripts/docs/doc_inventory.js

```
#!/usr/bin/env node
"use strict";

/**
```

```
 * scripts/docs/doc_inventory.js
 *
 * Produces a deterministic JSON report of repo documentation state by
reading MANIFEST.md files.
 *
 * Output (printed to stdout as JSON):
 * {
 *   "generated_at": "ISO8601",
 *   "repo_root": "<posix path>",
 *   "manifests": {
 *     "count": <int>,
 *     "paths": ["docs/.../MANIFEST.md", ...]
 *   },
 *   "expected": {
 *     "total": <int>,
 *     "by_folder": {
 *       "docs/..": {
 *         "manifest_path": "docs/.../MANIFEST.md",
 *         "expected_files": [{"name":"X.md","purpose":"..."}, ...]
 *       }
 *     }
 *   },
 *   "present": {
 *     "total_expected_present": <int>,
 *     "missing_total": <int>,
 *     "missing_by_folder": { "docs/...": ["A.md", ...] }
 *   },
 *   "orphans": {
 *     "files_not_in_any_manifest": ["docs/.../foo.md", ...],
 *     "folders_with_manifest_but_no_expected_files":
["docs/.../", ...]
 *   }
 * }
 *
 * Non-destructive. No external deps.
 * Exit code:
 * - 0 always (inventory is informational). If you want it to fail on
issues, wrap it in a gate script.
 */

const fs = require("fs");
const path = require("path");

const REPO_ROOT = process.cwd();
const MANIFEST_NAME = "MANIFEST.md";
const DEFAULT_ROOTS = ["docs", "scripts", "templates"]; // where to
look for manifests and/or doc files

function toPosix(p) {
  return p.split(path.sep).join("/");
```

```javascript
}

function isDir(p) {
  try {
    return fs.statSync(p).isDirectory();
  } catch {
    return false;
  }
}

function isFile(p) {
  try {
    return fs.statSync(p).isFile();
  } catch {
    return false;
  }
}

function readText(p) {
  return fs.readFileSync(p, "utf8");
}

function listDirsRecursive(startDir) {
  const out = [];
  const stack = [startDir];

  while (stack.length) {
    const cur = stack.pop();
    let entries;
    try {
      entries = fs.readdirSync(cur, { withFileTypes: true });
    } catch {
      continue;
    }

    for (const ent of entries) {
      const full = path.join(cur, ent.name);
      if (ent.isDirectory()) {
        out.push(full);
        stack.push(full);
      }
    }
  }

  return out;
}

function listFilesRecursive(startDir) {
  const out = [];
  const stack = [startDir];
  while (stack.length) {
    const cur = stack.pop();
    let entries;
    try {
      entries = fs.readdirSync(cur, { withFileTypes: true });
    } catch {
      continue;
    }

    for (const ent of entries) {
      const full = path.join(cur, ent.name);
      if (ent.isDirectory()) {
        stack.push(full);
      } else if (ent.isFile()) {
        out.push(full);
      }
    }
  }

  return out;
}

function findManifests() {
  const roots = DEFAULT_ROOTS.map((r) => path.join(REPO_ROOT,
r)).filter(isDir);
  const manifests = [];

  for (const root of roots) {
    const dirs = [root, ...listDirsRecursive(root)];
    for (const d of dirs) {
      const p = path.join(d, MANIFEST_NAME);
      if (fs.existsSync(p) && isFile(p)) manifests.push(p);
    }
  }

  // also include root MANIFEST.md if present
  const rootManifest = path.join(REPO_ROOT, MANIFEST_NAME);
  if (fs.existsSync(rootManifest) && isFile(rootManifest))
manifests.push(rootManifest);

  return Array.from(new Set(manifests))
    .sort((a, b) => toPosix(path.relative(REPO_ROOT,
a)).localeCompare(toPosix(path.relative(REPO_ROOT, b))));
}

function extractFolderLine(manifestText) {
  const m = manifestText.match(/^\*\*Folder:\*\*\s+`([^`]+)`\s*$/m);
  return m ? m[1] : null;
}
```

```
function extractExpectedFiles(manifestText) {
  const headerRe = /^## Expected Files\s*$/m;
  const match = manifestText.match(headerRe);
  if (!match) return { items: [], errors: ["Missing '## Expected
Files' section."] };

  const startIdx = manifestText.indexOf(match[0]) + match[0].length;
  const rest = manifestText.slice(startIdx);

  const nextHeading = rest.search(/^##\s+/m);
  const sectionBody = nextHeading === -1 ? rest : rest.slice(0,
nextHeading);

  const lines = sectionBody
    .split("\n")
    .map((l) => l.trim())
    .filter((l) => l.length > 0);

  const itemRe = /^-\s+`([^`]+)`\s+—\s+(.+)$/;

  const items = [];
  const errors = [];

  for (const line of lines) {
    const m = line.match(itemRe);
    if (!m) {
      errors.push(`Invalid Expected Files bullet format: "${line}"`);
      continue;
    }
    const name = m[1].trim();
    const purpose = m[2].trim();

    if (!name) errors.push(`Empty filename in Expected Files line: "$
{line}"`);
    if (!purpose) errors.push(`Empty purpose in Expected Files line:
"${line}"`);

    if (name.includes("/") || name.includes("\\") ||
name.includes("..")) {
      errors.push(`Invalid filename (must not include path separators
or ".."): "${name}"`);
    }

    items.push({ name, purpose });
  }

  return { items, errors };
}
```

```
function getAllRepoFilesUnderRoots() {
  const roots = DEFAULT_ROOTS.map((r) => path.join(REPO_ROOT,
r)).filter(isDir);
  const files = [];
  for (const root of roots) {
    files.push(...listFilesRecursive(root));
  }
  return files;
}

function main() {
  const manifests = findManifests();

  const expectedByFolder = {}; // folderRel -> { manifest_path,
expected_files[] }
  const expectedSet = new Set(); // absolute path of expected file
  const expectedMeta = new Map(); // abs expected path -> {folderRel,
name, purpose}

  const foldersWithManifestButNoExpected = [];
  const manifestParseWarnings = [];

  for (const manifestAbs of manifests) {
    const manifestRel = toPosix(path.relative(REPO_ROOT,
manifestAbs));
    const folderAbs = path.dirname(manifestAbs);
    const folderRel = toPosix(path.relative(REPO_ROOT, folderAbs));

    const text = readText(manifestAbs);

    // Folder line is informational in inventory, but we capture it to
highlight mismatch.
    const folderLine = extractFolderLine(text);
    if (folderLine && folderLine.replace(/\/+$/, "") !==
folderRel.replace(/\/+$/, "")) {
      manifestParseWarnings.push({
        manifest: manifestRel,
        warnings: [
          `Folder path mismatch: manifest says "${folderLine}" but
actual folder is "${folderRel}".`
        ]
      });
    }

    const { items, errors } = extractExpectedFiles(text);
    if (errors.length) {
      manifestParseWarnings.push({ manifest: manifestRel, warnings:
errors.slice() });
    }
    if (items.length === 0) {
      foldersWithManifestButNoExpected.push(folderRel.endsWith("/") ?
folderRel : folderRel + "/");
    }

    expectedByFolder[folderRel] = {
      manifest_path: manifestRel,
      expected_files: items.slice().sort((a, b) =>
a.name.localeCompare(b.name))
    };

    for (const it of items) {
      const absExpected = path.join(folderAbs, it.name);
      expectedSet.add(absExpected);
      expectedMeta.set(absExpected, { folderRel, name: it.name,
purpose: it.purpose });
    }
  }

  // Compute missing expected files
  const missingByFolder = {};
  let missingTotal = 0;
  let presentExpectedTotal = 0;

  for (const absExpected of Array.from(expectedSet)) {
    const meta = expectedMeta.get(absExpected);
    if (!fs.existsSync(absExpected)) {
      missingTotal += 1;
      if (!missingByFolder[meta.folderRel])
missingByFolder[meta.folderRel] = [];
      missingByFolder[meta.folderRel].push(meta.name);
    } else {
      presentExpectedTotal += 1;
    }
  }

  for (const k of Object.keys(missingByFolder)) {
    missingByFolder[k] = missingByFolder[k].slice().sort();
  }

  // Orphan files: files under roots not referenced by any manifest
expected list,
  // excluding MANIFEST.md itself.
  const allFiles = getAllRepoFilesUnderRoots()
    .filter((f) => isFile(f))
    .filter((f) => path.basename(f) !== MANIFEST_NAME);

  const orphans = [];
  for (const fAbs of allFiles) {
    if (!expectedSet.has(fAbs)) {
      orphans.push(toPosix(path.relative(REPO_ROOT, fAbs)));
    }
  }
  orphans.sort();

  const report = {
    generated_at: new Date().toISOString(),
    repo_root: toPosix(REPO_ROOT),
    manifests: {
      count: manifests.length,
      paths: manifests.map((p) => toPosix(path.relative(REPO_ROOT,
p))).sort()
    },
    expected: {
      total: expectedSet.size,
      by_folder: Object.fromEntries(
        Object.entries(expectedByFolder).sort((a, b) =>
a[0].localeCompare(b[0]))
      )
    },
    present: {
      total_expected_present: presentExpectedTotal,
      missing_total: missingTotal,
      missing_by_folder: Object.fromEntries(
        Object.entries(missingByFolder).sort((a, b) =>
a[0].localeCompare(b[0]))
      )
    },
    orphans: {
      files_not_in_any_manifest: orphans,
      folders_with_manifest_but_no_expected_files:
foldersWithManifestButNoExpected.sort()
    }
  };

  if (manifestParseWarnings.length) {
    report.manifest_warnings = manifestParseWarnings.sort((a, b) =>
a.manifest.localeCompare(b.manifest));
  }

  process.stdout.write(JSON.stringify(report, null, 2) + "\n");
}

main();
```

FILE: scripts/docs/find_missing_expected_files.js

```
#!/usr/bin/env node
"use strict";

/**
 * scripts/docs/find_missing_expected_files.js
 *
 * Reads MANIFEST.md files, extracts "Expected Files" entries,
 * then checks the filesystem to report which expected files are
missing.
 *
 * Non-destructive: read-only, prints report, exits non-zero if
missing found.
 * No external deps.
 */

const fs = require("fs");
const path = require("path");

const REPO_ROOT = process.cwd();
const MANIFEST_NAME = "MANIFEST.md";

function toPosix(p) {
  return p.split(path.sep).join("/");
}

function isDir(p) {
  try {
    return fs.statSync(p).isDirectory();
  } catch {
    return false;
  }
}

function readText(p) {
  return fs.readFileSync(p, "utf8");
}

function listDirsRecursive(startDir) {
  const out = [];
  const stack = [startDir];

  while (stack.length) {
    const cur = stack.pop();
    let entries;
    try {
      entries = fs.readdirSync(cur, { withFileTypes: true });
    } catch {
      continue;
    }

    for (const ent of entries) {
      const full = path.join(cur, ent.name);
      if (ent.isDirectory()) {
        out.push(full);
        stack.push(full);
      }
    }
  }
  return out;
}

function findManifests() {
  const roots = ["docs", "scripts", "templates"]
    .map((r) => path.join(REPO_ROOT, r))
    .filter(isDir);

  const manifests = [];

  for (const root of roots) {
    const dirs = [root, ...listDirsRecursive(root)];
    for (const d of dirs) {
      const p = path.join(d, MANIFEST_NAME);
      if (fs.existsSync(p)) manifests.push(p);
    }
  }

  const rootManifest = path.join(REPO_ROOT, MANIFEST_NAME);
  if (fs.existsSync(rootManifest)) manifests.push(rootManifest);

  return Array.from(new Set(manifests)).sort();
}

function extractExpectedFiles(manifestText) {
  const headerRe = /^## Expected Files\s*$/m;
  const match = manifestText.match(headerRe);
  if (!match) return { items: [], errors: ["Missing '## Expected
Files' section."] };

  const startIdx = manifestText.indexOf(match[0]) + match[0].length;
  const rest = manifestText.slice(startIdx);

  const nextHeading = rest.search(/^##\s+/m);
  const sectionBody = nextHeading === -1 ? rest : rest.slice(0,
nextHeading);

  const lines = sectionBody
    .split("\n")
    .map((l) => l.trim())
    .filter((l) => l.length > 0);

  const itemRe = /^-\s+(?:\[[ x]\]\s+)?`([^`]+)`\s+(?:—|-)\s+(.+)$/;

  const items = [];
  const errors = [];

  for (const line of lines) {
    const m = line.match(itemRe);
    if (!m) {
      errors.push(`Invalid Expected Files bullet format: "${line}"`);
      continue;
    }
    const name = m[1].trim();
    const purpose = m[2].trim();
    if (!name) errors.push(`Empty filename in Expected Files line: "$
{line}"`);
    if (!purpose) errors.push(`Empty purpose in Expected Files line:
"${line}"`);

    if (name.includes("/") || name.includes("\\") ||
name.includes("..")) {
      errors.push(`Invalid filename (must not include path separators
or ".."): "${name}"`);
    }

    items.push({ name, purpose });
  }

  return { items, errors };
}

function main() {
  const manifests = findManifests();

  if (manifests.length === 0) {
    console.log("No MANIFEST.md files found under docs/, scripts/,
templates/, or repo root.");
    process.exit(0);
  }

  let manifestsChecked = 0;
  let totalExpected = 0;
  let totalMissing = 0;

  // Group missing by folder for readability
  const missingByFolder = new Map();
  const manifestParseErrors = [];
  for (const manifestPath of manifests) {
    manifestsChecked += 1;
    const dirAbs = path.dirname(manifestPath);
    const dirRel = toPosix(path.relative(REPO_ROOT, dirAbs));

    const text = readText(manifestPath);
    const { items, errors } = extractExpectedFiles(text);

    if (errors.length) {
      manifestParseErrors.push({
        manifest: toPosix(path.relative(REPO_ROOT, manifestPath)),
        errors
      });
      // Still proceed with whatever items parsed correctly.
    }

    totalExpected += items.length;

    for (const item of items) {
      const expectedAbs = path.join(dirAbs, item.name);
      if (!fs.existsSync(expectedAbs)) {
        totalMissing += 1;
        if (!missingByFolder.has(dirRel)) missingByFolder.set(dirRel,
[]);
        missingByFolder.get(dirRel).push(item.name);
      }
    }
  }

  // Print parse errors first
  if (manifestParseErrors.length) {
    console.error("\nManifest parse warnings (format issues):");
    for (const m of manifestParseErrors) {
      console.error(`\n[WARN] ${m.manifest}`);
      for (const e of m.errors) console.error(`  - ${e}`);
    }
  }

  // Print missing expected files
  if (totalMissing > 0) {
    console.error("\nMissing expected files:");
    const folders = Array.from(missingByFolder.keys()).sort();
    for (const folder of folders) {
      console.error(`\n- ${folder}/`);
      const files = missingByFolder.get(folder).slice().sort();
      for (const f of files) console.error(`  - ${f}`);
    }

    console.error("\nSummary:");
    console.error(`manifests checked: ${manifestsChecked}`);
    console.error(`total expected files: ${totalExpected}`);
    console.error(`total missing files: ${totalMissing}`);
    process.exit(2);
  }

  console.log("No missing expected files.");
  console.log(`manifests checked: ${manifestsChecked}`);
  console.log(`total expected files: ${totalExpected}`);
}

main();
```

FILE: scripts/docs/link_check.js

```
#!/usr/bin/env node
"use strict";

/**
 * scripts/docs/link_check.js
 *
 * Checks internal Markdown links under docs/ (and optionally
scripts/templates if desired)
 * and reports broken links. No external deps.
 *
 * What it checks:
 * - Relative links: ./foo.md, ../bar.md, foo.md, subdir/file.md
 * - Root-ish links: /docs/... (treated as repo-root absolute)
 * - Ignores:
 *   - http/https/mailto/tel links
 *   - pure anchors: #section
 *   - images are treated same as links (still validated if local)
 *   - code blocks (fenced) are ignored to reduce false positives
 *
 * Anchor checking:
 * - If a link includes #anchor, it validates the target file exists
 * - It also validates the anchor exists by extracting headings in the
target Markdown
 *
 * Exit code:
 * - 0 if OK
 * - 2 if broken links found
 */

const fs = require("fs");
const path = require("path");

const REPO_ROOT = process.cwd();
const DEFAULT_ROOTS = ["docs"]; // add "scripts", "templates" if you
want

function toPosix(p) {
  return p.split(path.sep).join("/");
}

function isDir(p) {
  try {
    return fs.statSync(p).isDirectory();
  } catch {
    return false;
  }
}

function isFile(p) {
  try {
    return fs.statSync(p).isFile();
  } catch {
    return false;
  }
}

function readText(p) {
  return fs.readFileSync(p, "utf8");
}

function listFilesRecursive(startDir) {
  const out = [];
  const stack = [startDir];

  while (stack.length) {
    const cur = stack.pop();
    let entries;
    try {
      entries = fs.readdirSync(cur, { withFileTypes: true });
    } catch {
      continue;
    }

    for (const ent of entries) {
      const full = path.join(cur, ent.name);
      if (ent.isDirectory()) {
        stack.push(full);
      } else if (ent.isFile()) {
        out.push(full);
      }
    }
  }
```

```javascript
  return out;
}

function stripFencedCodeBlocks(md) {
  // Remove ```...``` blocks (best-effort). Keeps line count
irrelevant, we only care about links.
  return md.replace(/```[\s\S]*?```/g, "");
}

function isExternalLink(url) {
  const u = url.toLowerCase();
  return (
    u.startsWith("http://") ||
    u.startsWith("https://") ||
    u.startsWith("mailto:") ||
    u.startsWith("tel:")
  );
}

function splitLinkTarget(raw) {
  // raw may include URL encoding; keep as-is but strip surrounding
whitespace.
  const t = raw.trim();
  const hashIdx = t.indexOf("#");
  if (hashIdx === -1) return { filePart: t, anchorPart: null };
  return {
    filePart: t.slice(0, hashIdx),
    anchorPart: t.slice(hashIdx + 1) || ""
  };
}

function normalizeRepoAbsolute(filePart) {
  // Treat "/docs/..." as repo-root absolute
  if (filePart.startsWith("/")) return filePart.slice(1);
  return null;
}

function decodeMaybe(s) {
  try {
    return decodeURIComponent(s);
  } catch {
    return s;
  }
}

function resolveTargetPath(fromFileAbs, filePart) {
  const decoded = decodeMaybe(filePart);

  // empty means same file (anchor-only handled elsewhere)
  if (!decoded) return null;
```

```
  const repoAbs = normalizeRepoAbsolute(decoded);
  if (repoAbs !== null) {
    return path.join(REPO_ROOT, repoAbs);
  }

  // Standard relative resolution from file directory
  const fromDir = path.dirname(fromFileAbs);
  return path.resolve(fromDir, decoded);
}

function slugifyHeading(text) {
  // GitHub-ish anchor behavior (approx):
  // - lowercase
  // - trim
  // - remove punctuation
  // - spaces to hyphens
  // - collapse hyphens
  let s = text.toLowerCase().trim();

  // Remove inline code ticks
  s = s.replace(/`+/g, "");

  // Remove punctuation except spaces and hyphens
  s = s.replace(/[^\w\s-]/g, "");

  // Spaces to hyphens
  s = s.replace(/\s+/g, "-");

  // Collapse multiple hyphens
  s = s.replace(/-+/g, "-");

  return s;
}

function extractAnchorsFromMarkdown(md) {
  // Collect anchors generated from headings: #, ##, ### ...
  // Also track duplicates by appending "-n" like GitHub does
(approx).
  const lines = md.split("\n");
  const baseCounts = new Map();
  const anchors = new Set();

  for (const line of lines) {
    const m = line.match(/^(#{1,6})\s+(.+?)\s*$/);
    if (!m) continue;

    let title = m[2];
    // Strip trailing hashes like "## Title ##"
    title = title.replace(/\s+#+\s*$/, "").trim();

    const base = slugifyHeading(title);
    if (!base) continue;

    const count = baseCounts.get(base) || 0;
    baseCounts.set(base, count + 1);

    const anchor = count === 0 ? base : `${base}-${count}`;
    anchors.add(anchor);
  }

  return anchors;
}

const anchorCache = new Map(); // absPath -> Set(anchors)

function getAnchorsForFile(absPath) {
  if (anchorCache.has(absPath)) return anchorCache.get(absPath);
  if (!isFile(absPath)) {
    anchorCache.set(absPath, new Set());
    return anchorCache.get(absPath);
  }
  const text = readText(absPath);
  const md = stripFencedCodeBlocks(text);
  const anchors = extractAnchorsFromMarkdown(md);
  anchorCache.set(absPath, anchors);
  return anchors;
}

function findMarkdownLinks(md) {
  // Finds [text](target) and ![alt](target)
  // Skips image alt/text content, just extracts the target.
  // Does not attempt to parse nested parentheses robustly (good
enough for specs).
  const links = [];
  const re = /!?\[[^\]]*?\]\(([^)]+?)\)/g;
  let m;
  while ((m = re.exec(md)) !== null) {
    links.push({ rawTarget: m[1], index: m.index });
  }
  return links;
}

function shouldIgnoreTarget(t) {
  const s = t.trim();
  if (!s) return true;
  // ignore pure anchors (handled elsewhere with filePart empty)
  if (s.startsWith("#")) return false;

  if (isExternalLink(s)) return true;

  // ignore common non-file schemes
  const lower = s.toLowerCase();
  if (lower.startsWith("data:")) return true;

  return false;
}

function main() {
  const roots = DEFAULT_ROOTS
    .map((r) => path.join(REPO_ROOT, r))
    .filter(isDir);

  if (roots.length === 0) {
    console.log(`No roots found to scan: ${DEFAULT_ROOTS.join(", ")}
`);
    process.exit(0);
  }

  const mdFiles = [];
  for (const root of roots) {
    for (const f of listFilesRecursive(root)) {
      if (f.toLowerCase().endsWith(".md")) mdFiles.push(f);
    }
  }

  if (mdFiles.length === 0) {
    console.log("No Markdown files found.");
    process.exit(0);
  }

  const broken = [];

  for (const fileAbs of mdFiles) {
    const relFile = toPosix(path.relative(REPO_ROOT, fileAbs));
    const raw = readText(fileAbs);
    const md = stripFencedCodeBlocks(raw);

    const links = findMarkdownLinks(md);

    for (const link of links) {
      const target = link.rawTarget;

      if (shouldIgnoreTarget(target)) continue;
      const { filePart, anchorPart } = splitLinkTarget(target);

      // Anchor-only link within same file: (#something)
      if (!filePart && anchorPart !== null) {
        const anchors = getAnchorsForFile(fileAbs);
        const wanted = decodeMaybe(anchorPart).toLowerCase();
        if (wanted && !anchors.has(wanted)) {
          broken.push({
            from: relFile,
            to: target.trim(),
            reason: `Missing anchor "#${wanted}" in same file`
          });
        }
        continue;
      }

      const resolvedAbs = resolveTargetPath(fileAbs, filePart);
      if (!resolvedAbs) continue;

      // If they linked to a directory, consider it broken (markdown
links should target a file)
      if (fs.existsSync(resolvedAbs) && isDir(resolvedAbs)) {
        broken.push({
          from: relFile,
          to: target.trim(),
          reason: "Link points to a directory, not a file"
        });
        continue;
      }

      if (!fs.existsSync(resolvedAbs) || !isFile(resolvedAbs)) {
        broken.push({
          from: relFile,
          to: target.trim(),
          reason: `Target file not found: $
{toPosix(path.relative(REPO_ROOT, resolvedAbs))}`
        });
        continue;
      }

      // Anchor validation (only for markdown targets)
      if (anchorPart !== null && anchorPart.length > 0 &&
resolvedAbs.toLowerCase().endsWith(".md")) {
        const anchors = getAnchorsForFile(resolvedAbs);
        const wanted = decodeMaybe(anchorPart).toLowerCase();
        if (!anchors.has(wanted)) {
          broken.push({
            from: relFile,
            to: target.trim(),
            reason: `Missing anchor "#${wanted}" in target`
          });
        }
      }
    }
  }

  if (broken.length > 0) {
    console.error(`Broken links found: ${broken.length}\n`);
    for (const b of broken) {
      console.error(`[FAIL] ${b.from}`);
      console.error(`  -> ${b.to}`);
      console.error(`  reason: ${b.reason}\n`);
    }
    process.exit(2);
  }

  console.log(`Checked ${mdFiles.length} markdown file(s).`);
  console.log("No broken links found.");
}

main();
```

FILE: scripts/docs/link_check.py

```
# link_check.py
# Checks for broken internal markdown links.

import re

def find_md_links():
    # Simple regex for [text](link)
    re.compile(r'\[.+?\]\((.+?)\)')
    # Placeholder for link checking logic
    print("Checking markdown links...")
    print("All links verified (Mock).")

if __name__ == "__main__":
    find_md_links()
```

FILE: scripts/docs/toc_builder.py

```
# toc_builder.py
# Automatically generates Table of Contents for large markdown files.

import sys

def build_toc(filename):
    print(f"Generating TOC for {filename}...")
    # Placeholder for TOC generation logic

if __name__ == "__main__":
    if len(sys.argv) > 1:
        build_toc(sys.argv[1])
    else:
        print("Usage: python toc_builder.py <file.md>")
```

FILE: scripts/docs/toc_sync.js

```
#!/usr/bin/env node
"use strict";

/**
 * scripts/docs/toc_sync.js
 *
 * Deterministically syncs curated TOC/index docs from MANIFEST.md
"Expected Files".
 *
 * Strategy:
 * - Read MANIFEST.md files under docs/
 * - For selected index targets, replace the content between:
 *     <!-- AUTO-GENERATED:START -->
 *     <!-- AUTO-GENERATED:END -->
 *   with a generated list derived from a source folder's manifest.
 *
 * This script:
 * - Overwrites ONLY the auto-generated block in each target index
file
 * - Does NOT create missing index files
 * - Does NOT create any placeholder docs
 * - Uses built-in modules only
 *
 * Exit code:
 * - 0 on success
 * - 2 if any target file is missing, or required manifest is
missing/invalid
 */

const fs = require("fs");
const path = require("path");

const REPO_ROOT = process.cwd();

const AUTO_START = "<!-- AUTO-GENERATED:START -->";
const AUTO_END = "<!-- AUTO-GENERATED:END -->";
```

```
// Index targets to sync.
// Add more entries as you add curated index files.
const INDEX_TARGETS = [
  {
    target: "docs/02_DESIGN/UX_SPECS/SCREEN_INDEX.md",
    title: "Screen Specs",
    sourceFolder: "docs/02_DESIGN/UX_SPECS/SCREEN_SPECS",
    linkPrefix: "./SCREEN_SPECS", // links in SCREEN_INDEX are
relative to its folder
    description:
      "This index is generated from the SCREEN_SPECS folder manifest.
Edit only outside the auto-generated block."
  }
];

function toPosix(p) {
  return p.split(path.sep).join("/");
}

function isFile(p) {
  try {
    return fs.statSync(p).isFile();
  } catch {
    return false;
  }
}

function isDir(p) {
  try {
    return fs.statSync(p).isDirectory();
  } catch {
    return false;
  }
}

function readText(p) {
  return fs.readFileSync(p, "utf8");
}

function writeText(p, s) {
  fs.writeFileSync(p, s, "utf8");
}

function listDirsRecursive(startDir) {
  const out = [];
  const stack = [startDir];

  while (stack.length) {
    const cur = stack.pop();
    let entries;
    try {
      entries = fs.readdirSync(cur, { withFileTypes: true });
    } catch {
      continue;
    }
    for (const ent of entries) {
      const full = path.join(cur, ent.name);
      if (ent.isDirectory()) {
        out.push(full);
        stack.push(full);
      }
    }
  }
  return out;
}

function findManifestForFolder(folderRelPosix) {
  const folderAbs = path.join(REPO_ROOT,
...folderRelPosix.split("/"));
  const manifestAbs = path.join(folderAbs, "MANIFEST.md");
  return isFile(manifestAbs) ? manifestAbs : null;
}

function extractExpectedFiles(manifestText) {
  const headerRe = /^## Expected Files\s*$/m;
  const match = manifestText.match(headerRe);
  if (!match) return { items: [], errors: ["Missing '## Expected
Files' section."] };

  const startIdx = manifestText.indexOf(match[0]) + match[0].length;
  const rest = manifestText.slice(startIdx);

  const nextHeading = rest.search(/^##\s+/m);
  const sectionBody = nextHeading === -1 ? rest : rest.slice(0,
nextHeading);

  const lines = sectionBody
    .split("\n")
    .map((l) => l.trim())
    .filter((l) => l.length > 0);

  const itemRe = /^-\s+`([^`]+)`\s+—\s+(.+)$/;

  const items = [];
  const errors = [];

  for (const line of lines) {
    const m = line.match(itemRe);
    if (!m) {
      errors.push(`Invalid Expected Files bullet format: "${line}"`);
      continue;
    }
    const name = m[1].trim();
    const purpose = m[2].trim();

    if (!name) errors.push(`Empty filename in Expected Files line: "$
{line}"`);
    if (!purpose) errors.push(`Empty purpose in Expected Files line:
"${line}"`);
    if (name.includes("/") || name.includes("\\") ||
name.includes("..")) {
      errors.push(`Invalid filename (must not include path separators
or ".."): "${name}"`);
      continue;
    }

    items.push({ name, purpose });
  }

  return { items, errors };
}

function ensureAutoBlockExists(text, targetRel) {
  const startIdx = text.indexOf(AUTO_START);
  const endIdx = text.indexOf(AUTO_END);

  if (startIdx === -1 || endIdx === -1 || endIdx < startIdx) {
    return {
      ok: false,
      error:
        `Missing or malformed auto-generated markers in ${targetRel}.
` +
        `Add these lines:\n${AUTO_START}\n${AUTO_END}`
    };
  }
  return { ok: true };
}

function replaceAutoBlock(text, replacement) {
  const startIdx = text.indexOf(AUTO_START);
  const endIdx = text.indexOf(AUTO_END);
  const before = text.slice(0, startIdx + AUTO_START.length);
  const after = text.slice(endIdx);

  // Ensure single blank line around content for stable formatting.
  return `${before}\n\n${replacement.trimEnd()}\n\n${after}`;
}

function buildListMarkdown({ title, description, items, linkPrefix })
{
  // Deterministic: sort by filename.
  const sorted = items.slice().sort((a, b) =>
a.name.localeCompare(b.name));

  // Link path uses POSIX separators
  const lines = [];
  lines.push(`### ${title}`);
  lines.push("");
  lines.push(description);
  lines.push("");
  for (const it of sorted) {
    const link = `${linkPrefix.replace(/\/+$/, "")}/${it.name}`;
    lines.push(`- [\`${it.name}\`](${link}) — ${it.purpose}`);
  }
  return lines.join("\n");
}

function main() {
  const docsRoot = path.join(REPO_ROOT, "docs");
  if (!isDir(docsRoot)) {
    console.error("docs/ folder not found. Nothing to sync.");
    process.exit(0);
  }

  let failures = 0;
  let updated = 0;
  let skipped = 0;

  for (const target of INDEX_TARGETS) {
    const targetAbs = path.join(REPO_ROOT,
...target.target.split("/"));
    const targetRel = target.target;

    if (!isFile(targetAbs)) {
      failures += 1;
      console.error(`[FAIL] Target index file missing: ${targetRel}`);
      continue;
    }

    const manifestAbs = findManifestForFolder(target.sourceFolder);
    if (!manifestAbs) {
      failures += 1;
      console.error(
        `[FAIL] Required manifest missing for source folder: $
{target.sourceFolder}/MANIFEST.md`
      );
      continue;
    }

    const manifestText = readText(manifestAbs);
    const parsed = extractExpectedFiles(manifestText);
    if (parsed.errors.length) {
      failures += 1;
      console.error(`[FAIL] Manifest parse errors in $
{toPosix(path.relative(REPO_ROOT, manifestAbs))}`);
      for (const e of parsed.errors) console.error(`  - ${e}`);
      continue;
    }

    const indexText = readText(targetAbs);

    const markerCheck = ensureAutoBlockExists(indexText, targetRel);
    if (!markerCheck.ok) {
      failures += 1;
      console.error(`[FAIL] ${markerCheck.error}`);
      continue;
    }

    const replacement = buildListMarkdown({
      title: target.title,
      description: target.description,
      items: parsed.items,
      linkPrefix: target.linkPrefix
    });

    const nextText = replaceAutoBlock(indexText, replacement);

    if (nextText !== indexText) {
      writeText(targetAbs, nextText);
      updated += 1;
      console.log(`[OK] Updated: ${targetRel}`);
    } else {
      skipped += 1;
      console.log(`[OK] No changes: ${targetRel}`);
    }
  }

  console.log(`targets: ${INDEX_TARGETS.length}`);
  console.log(`updated: ${updated}`);
  console.log(`unchanged: ${skipped}`);
  console.log(`failures: ${failures}`);

  if (failures > 0) process.exit(2);
}

main();
```

FILE: templates/doc_templates/TEMPLATE_ADR.md

[ADR-0000]: [Title]

Status

[PROPOSED | ACCEPTED | SUPERSEDED]

Context

[What is the problem we are solving?]

Decision

[What did we decide?]

Rationale

[Why is this the best path?]

FILE: templates/doc_templates/TEMPLATE_API.md

API Specification: [Endpoint Name]

Overview

[What does this API do?]

Request

```
[METHOD] [PATH]
```

Headers

- `Authorization`: Bearer token

Parameters

[Query or Body parameters]

Response

```json
{
  "status": "success",
  "data": {}
}
```

FILE: templates/doc_templates/TEMPLATE_SCREEN_SPEC.md

Screen Spec: [Screen Name]

Visual Design

- **Key Palette**: [Nebula Colors]
- **Glassmorphism**: [Blur Radius]

Interaction

1. [User Step 1]
2. [User Step 2]

Data Requirements

- [Entity 1]
- [Entity 2]

FILE: templates/doc_templates/TEMPLATE_SPEC.md

Specification: [Feature Name]

Purpose

[Detailed goal of this feature]

Requirements

- [Requirement 1]
- [Requirement 2]

Constraints

- [Constraint 1]

FILE: templates/doc_templates/TEMPLATE_TEST_CASE.md

Test Case: [Test Name]

Objective

[What are we testing?]

Steps

1. [Step 1]
2. [Step 2]

Expected Result

[Success criteria]

DESIGN AND UX SPECIFICATION

Generated on: 2026-01-30T16:22:34.949Z

FILE: docs/02_DESIGN/CONTENT_STYLE_GUIDE.md

Content Style Guide

Voice, tone, and grammar rules for the Memoir.ai interface.

1. The Voice: "The Sage Observer"

The UI should sound empathetic but objective. Avoid overly "Chipper" or "Corporate" language.

- **DO**: "Your vault is ready for exploration."
- **DON'T**: "Yay! You're all set to go!"

2. Key Terminology

- **Vault**: The local, encrypted storage unit. (Capitalize as proper noun).
- **Snapshot**: A generated AI narrative.
- **Evidence**: Raw messages or photos used to support a claim.
- **Timeline**: The scrollable unified feed.

3. Error Message Guidelines

Errors should follow the **Nebula-Response** pattern:

1. State what happened (Neutral).
2. Explain the impact.
3. Provide a clear "Next Step".

4. Formatting

- Use **Title Case** for primary navigation items.
- Use **Sentence case** for descriptions and body text.

FILE: docs/02_DESIGN/DESIGN_SYSTEM/COLOR_SYSTEM.md

Color System

The "Nebula" palette. Emphasizing depth, luminosity, and high contrast.

1. Primary Palette

- **Obsidian (Base)**: `#050505` - Main background.
- **Star-Violet**: `#8B5CF6` - Primary action color.
- **Nebula-Cyan**: `#06B6D4` - Secondary/Teal accent.
- **Supernova-Magenta**: `#D946EF` - Highlights and alerts.

2. Dynamic States

- **Hover**: 20% increased luminosity of the base accent.
- **Active**: 10% decreased luminosity + subtle bloom effect.

3. UI Backgrounds

- **Glass-Base**: `rgba(255, 255, 255, 0.03)` with `backdrop-filter: blur(20px)`.
- **Frost-Edge**: `rgba(255, 255, 255, 0.1)` border.

FILE: docs/02_DESIGN/DESIGN_SYSTEM/COMPONENT_INVENTORY.md

Component Inventory — Memoir.ai V1

This document lists the core UI components required for Memoir.ai V1, as specified in the Product Requirements and Build Spec.

1. Foundation Components

Button

- **Variants**: Primary, Secondary, Ghost, Destructive.
- **States**: Default, Hover, Active, Disabled, Loading.
- **Usage**: Primary actions like "Generate Snapshot" or "Import Data".

Modal

- **Props**: `title`, `children`, `onClose`, `isOpen`.
- **Usage**: Import wizards, confirmation dialogs (Library deletion), and settings.

ProgressBar

- **Props**: `percentage`, `status` (pending, running, complete, failed).
- **Usage**: Visualizing the progress of background import/export jobs.

2. Business Components

TimelineItem

- **Props**: `timestamp`, `source_icon`, `content_snippet`.
- **Aesthetic**: High information density, easy for scanning large datasets.

TimelineGroup

- **Props**: `date`, `children` (TimelineItems).
- **Logic**: Groups memories by date (e.g., "Mondy, Jan 26").

CitationComponent

- **Props**: `citation_id`, `source_memory_id`, `text`.
- **Aesthetic**: Non-intrusive "pill" or numbered marker within a Snapshot.
- **Interaction**: Click to scroll/jump to the source memory in the timeline.

3. Navigation & Search

Global Search Bar

- **Features**: Real-time filtering, syntax support (source:messenger), and date range pickers.
- **Placement**: Historically fixed at the top of the Library workspace.

Sidebar

- **Sections**: Library switcher, Timeline, Search, Snapshots, Settings.

FILE: docs/02_DESIGN/DESIGN_SYSTEM/DESIGN_PRINCIPLES.md

Design Principles

1. **Local-First, Zero-Exfiltration**: The UI must never leak data status to the cloud without intent.
2. **Cinematic Immersion**: Use motion and depth (z-layers) to make the archival process feel like discovery, not homework.
3. **Typography as Structure**: Minimize lines; use whitespace and weight hierarchy to define zones.
4. **The "Evidence First" Rule**: No AI generated content should be shown without its citation "Anchor" visible or reachable.

FILE: docs/02_DESIGN/DESIGN_SYSTEM/ICONOGRAPHY.md

Iconography

Icons in Memoir.ai serve as **navigational beacons** in the void. They must be precise, lightweight, and clear.

1. Style Guide

- **Stroke-Based**: We use a clean stroke style (1.5px to 2px). No solid fills unless representing a toggled state or a specific brand asset.
- **Geometry**: Icons should feel "engineered." Perfect circles, straight lines, distinct corners.
- **Open Contours**: Avoid heavy, closed shapes. The icons should feel like they are made of light (neon tubes).

2. Recommended Set

We use **Lucide React** (or Heroicons Outline) as the standard library.

- **Consistency**: Adhere to the standard SVG stroke width.
- **Corners**: Rounded joins (`stroke-linejoin="round"`) match the UI's border radius.

3. Sizing

Token	Size	Usage
Small	16px	Inside dense buttons, meta-data rows, timestamps.
Medium	20px	Navigation items, standard inputs, table actions.
Large	24px	Section headers, empty states, major toggles.

4. Color & State

- **Neutral**: By default, icons inherit the text color (`currentColor` or `--ink-2`).
- **Active/Glow**:
  - Active navigation items often glow `--cyan`.
  - Destructive actions glow `--danger`.
- **Stroke**: Ensure `vector-effect: non-scaling-stroke` if resizing heavily (though we prefer swapping size variants).

5. Usage Examples

```
/* DO */
<Icon name="search" size={20} color="var(--ink-2)" />
```

```
/* DON'T */
/* Filled icons for non-active states */
<Icon name="search-filled" />
```

FILE: docs/02_DESIGN/DESIGN_SYSTEM/SPACING_ELEVATION.md

Spacing & Elevation

Memoir.ai uses a fluid spacing system and a multi-layered glass elevation model to create depth in the void.

1. Spacing System

We do not use fixed pixels for structural padding. We use **Fluid Spacing Tokens** that adapt to the viewport width using `clamp()`. This ensures the interface never feels cramped on laptops or sparse on large monitors.

Token	CSS Variable	Value Range (approx)	Usage
Small	`--pad-sm`	`10px`	Internal component padding (buttons, inputs).
Medium	`--pad-md`	`16px`	Standard gap between small elements.
Large	`--pad-lg`	`16px - 24px`	Card padding, section gaps.
Extra Large	`--pad-xl`	`20px - 32px`	Page margins, major section breaks.

The Grid

While fluid, we align to a **4px baseline grid** where possible.

- Elements are spaced by `4px`, `8px`, `16px`, `24px`, `32px`, `48px`, `64px`.

2. Elevation & Depth

In the "Cosmic Glass" aesthetic, elevation is defined by **opacity** and **blur**, not just shadow.

Layer 0: The Void

- **Variable**: `--bg-0`
- **Use**: The infinite background. Never place content directly here; use a panel.

Layer 1: Base Panel

- **Variable**: `--panel` (`rgba(27, 16, 61, 0.45)`)
- **Use**: Sidebar, large background containers.
- **Effect**: Low opacity, high blur. Recedes behind content.

Layer 2: Standard Glass

- **Variable**: `--panel-2` (`rgba(19, 12, 46, 0.55)`)
- **Use**: Cards, feed items, tiles.
- **Effect**: Medium opacity, medium blur. The distinct "objects" in the space.

Layer 3: Floating / Hard Glass

- **Variable**: `--panel-3` (`rgba(12, 8, 32, 0.75)`)
- **Use**: Modals, dropdowns, sticky headers.
- **Effect**: High opacity, crisp border, bright specular highlight.

3. Shadows

Shadows represent the light cast by the UI elements.

- **`--shadow-lg`**: Deep, ambient shadow for floating elements.
  - `0 24px 60px rgba(0,0,0,0.55)`
- **`--shadow-glow`**: Neon diffusion for active states.
  - `0 0 40px rgba(123, 97, 255, 0.15)`

FILE: docs/02_DESIGN/DESIGN_SYSTEM/THEME.css

```css
/* theme.css — Memoir.ai “cosmic glass” design system
   One CSS file to rule many HTMLs (tragically useful).

   Goals:
   - Deep indigo/near-black base
   - Purple/violet glass panels with soft neon edges
   - Cyan → violet → magenta accent gradients
   - Consistent typography + spacing
   - Document-friendly + UI-friendly components
*/

/* ========== 0) Design Tokens ========== */
:root {
    /* Typography */
    --font-sans: ui-sans-serif, system-ui, -apple-system, "Segoe UI",
Roboto, Helvetica, Arial;
    --font-mono: ui-monospace, SFMono-Regular, Menlo, Monaco,
Consolas, "Liberation Mono", "Courier New";

    /* Core palette (pulled from your screenshots/icon palette) */
    --bg-0: #040313;
    /* near-black indigo */
    --bg-1: #070521;
    /* deep night */
    --bg-2: #0B0830;
    /* deep purple */
    --ink-0: #F5F3FF;
    /* near-white */
    --ink-1: rgba(245, 243, 255, .88);
    --ink-2: rgba(245, 243, 255, .68);
    --ink-3: rgba(245, 243, 255, .48);

    /* Surfaces (glass) */
    --panel: rgba(27, 16, 61, .45);
    --panel-2: rgba(19, 12, 46, .55);
    --panel-3: rgba(12, 8, 32, .70);

    /* Borders / separators */
    --stroke: rgba(210, 185, 255, .20);
    --stroke-2: rgba(210, 185, 255, .12);
    --hairline: rgba(255, 255, 255, .08);

    /* Neon accents */
    --violet: #7B61FF;
    --indigo: #4C3397;
    --blue: #6CB7FF;
    --cyan: #7FE7FF;
    --magenta: #FF4FD8;
    --pink: #F08BFF;

    /* Status */
    --ok: #3BE8C3;
    --warn: #FFB547;
    --danger: #FF5A6A;

    /* Effects */
    --shadow-lg: 0 24px 60px rgba(0, 0, 0, .55);
    --shadow-md: 0 14px 34px rgba(0, 0, 0, .45);
    --shadow-sm: 0 8px 18px rgba(0, 0, 0, .35);

    --blur: 18px;
    --radius-lg: 18px;
    --radius-md: 14px;
    --radius-sm: 12px;
    --pad-xl: 28px;
    --pad-lg: 22px;
    --pad-md: 16px;
    --pad-sm: 12px;

    --page-max: 980px;

    /* Gradients */
    --grad-accent: linear-gradient(90deg, rgba(127, 231, 255, .85),
rgba(123, 97, 255, .85), rgba(255, 79, 216, .85));
    --grad-accent-soft: linear-gradient(90deg, rgba(127, 231, 255,
.22), rgba(123, 97, 255, .22), rgba(255, 79, 216, .22));
    --grad-surface: linear-gradient(180deg, rgba(255, 255, 255, .08),
rgba(255, 255, 255, .02));
    --grad-edge: linear-gradient(180deg, rgba(255, 255, 255, .18),
rgba(255, 255, 255, .02));

    color-scheme: dark;
}

/* ========== 1) Base Reset ========== */
* {
    box-sizing: border-box;
}

html,
body {
    height: 100%;
}

body {
    margin: 0;
    font-family: var(--font-sans);
    line-height: 1.55;
    color: var(--ink-1);
    background:
        radial-gradient(1200px 700px at 75% 25%, rgba(255, 79, 216,
.14), transparent 60%),
        radial-gradient(900px 600px at 15% 70%, rgba(127, 231, 255,
.10), transparent 55%),
        radial-gradient(1100px 800px at 50% 50%, rgba(123, 97, 255,
.14), transparent 60%),
        linear-gradient(180deg, var(--bg-0), var(--bg-1) 50%, var(--
bg-2));
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}

img {
    max-width: 100%;
```

```css
    height: auto;
}

a {
    color: inherit;
    text-decoration: none;
}

a:hover {
    text-decoration: underline;
    text-decoration-color: rgba(255, 255, 255, .25);
}

code,
pre {
    font-family: var(--font-mono);
}

::selection {
    background: rgba(123, 97, 255, .35);
    color: var(--ink-0);
}

hr {
    border: 0;
    height: 1px;
    background: linear-gradient(90deg, transparent, rgba(210, 185,
255, .25), transparent);
    margin: 18px 0;
}

/* ========== 2) Layout Helpers ========== */
.container {
    width: min(var(--page-max), calc(100% - 2*var(--pad-xl)));
    margin: 0 auto;
}

.stack {
    display: flex;
    flex-direction: column;
    gap: var(--pad-md);
}

.row {
    display: flex;
    gap: var(--pad-md);
    flex-wrap: wrap;
}
```

```css
.center {
    display: grid;
    place-items: center;
}

.muted {
    color: var(--ink-2);
}

.small {
    font-size: 12px;
    color: var(--ink-2);
}

.kbd {
    font-family: var(--font-mono);
    font-size: 12px;
    padding: 2px 8px;
    border-radius: 10px;
    background: rgba(255, 255, 255, .06);
    border: 1px solid rgba(255, 255, 255, .10);
    color: var(--ink-1);
}

/* ========== 3) Glass Surfaces ========== */
.glass {
    background: var(--panel);
    border: 1px solid var(--stroke-2);
    border-radius: var(--radius-lg);
    box-shadow: var(--shadow-md);
    backdrop-filter: blur(var(--blur));
    -webkit-backdrop-filter: blur(var(--blur));
}

.glass.soft {
    background: var(--panel-2);
    border-color: rgba(210, 185, 255, .14);
    box-shadow: var(--shadow-sm);
}

.glass.hard {
    background: var(--panel-3);
    border-color: rgba(255, 255, 255, .10);
}

.glass.edge-glow {
    position: relative;
    overflow: hidden;
}
```

```
.glass.edge-glow::before {
    content: "";
    position: absolute;
    inset: 0;
    border-radius: inherit;
    padding: 1px;
    background: var(--grad-accent-soft);
    -webkit-mask:
        linear-gradient(#000 0 0) content-box,
        linear-gradient(#000 0 0);
    -webkit-mask-composite: xor;
    mask-composite: exclude;
    opacity: .85;
    pointer-events: none;
}

.glass.edge-glow::after {
    content: "";
    position: absolute;
    inset: -40px -60px auto auto;
    width: 220px;
    height: 140px;
    background: radial-gradient(circle at 30% 30%, rgba(255, 79, 216,
.22), transparent 70%);
    filter: blur(10px);
    opacity: .8;
    pointer-events: none;
}

/* ========== 4) App Shell (matches your UI screenshots) ========== */
.app {
    min-height: 100vh;
    display: grid;
    grid-template-columns: 260px 1fr;
    gap: var(--pad-lg);
    padding: var(--pad-xl);
}

.sidebar {
    padding: var(--pad-lg);
    border-radius: var(--radius-lg);
}

.sidebar.glass {
    background: rgba(16, 10, 40, .55);
}

.brand {
    display: flex;
    align-items: center;
    gap: 12px;
    padding: 8px 6px 18px;
}

.brand .logo {
    width: 34px;
    height: 34px;
    border-radius: 12px;
    background: var(--grad-accent);
    box-shadow: 0 10px 26px rgba(123, 97, 255, .28);
}

.brand .name {
    font-weight: 650;
    letter-spacing: .2px;
    color: var(--ink-0);
}

.nav {
    display: flex;
    flex-direction: column;
    gap: 6px;
    padding-top: 8px;
}

.nav a {
    display: flex;
    align-items: center;
    gap: 10px;
    padding: 10px 12px;
    border-radius: 12px;
    color: var(--ink-2);
    border: 1px solid transparent;
    background: transparent;
}

.nav a:hover {
    color: var(--ink-0);
    background: rgba(255, 255, 255, .04);
    border-color: rgba(255, 255, 255, .08);
}

.nav a.active {
    color: var(--ink-0);
    background: rgba(123, 97, 255, .14);
    border-color: rgba(123, 97, 255, .25);
    box-shadow: 0 10px 26px rgba(123, 97, 255, .12);
}

.nav .icon {
```

```css
    width: 18px;
    height: 18px;
    opacity: .9;
}

.main {
    display: flex;
    flex-direction: column;
    gap: var(--pad-lg);
    min-width: 0;
}

.topbar {
    padding: 14px 16px;
    border-radius: var(--radius-lg);
    display: flex;
    align-items: center;
    gap: 12px;
    justify-content: space-between;
}

.search {
    flex: 1;
    max-width: 520px;
}

/* ========== 5) Typography ========== */
h1,
h2,
h3 {
    margin: 0;
    color: var(--ink-0);
    letter-spacing: .2px;
}

h1 {
    font-size: 30px;
    font-weight: 720;
}

h2 {
    font-size: 18px;
    font-weight: 680;
}

h3 {
    font-size: 15px;
    font-weight: 650;
}
```

```css
p {
    margin: 0;
    color: var(--ink-1);
}

.lede {
    color: var(--ink-2);
    max-width: 72ch;
}

/* ========== 6) Buttons ========== */
.btn {
    appearance: none;
    border: 1px solid rgba(255, 255, 255, .12);
    background: rgba(255, 255, 255, .06);
    color: var(--ink-0);
    padding: 10px 14px;
    border-radius: 12px;
    font-weight: 600;
    letter-spacing: .2px;
    cursor: pointer;
    box-shadow: 0 10px 22px rgba(0, 0, 0, .22);
    backdrop-filter: blur(10px);
    -webkit-backdrop-filter: blur(10px);
    transition: transform .12s ease, border-color .12s ease,
background .12s ease, box-shadow .12s ease;
}

.btn:hover {
    background: rgba(255, 255, 255, .08);
    border-color: rgba(210, 185, 255, .22);
    box-shadow: 0 14px 30px rgba(0, 0, 0, .30);
}

.btn:active {
    transform: translateY(1px);
}

.btn.primary {
    border-color: rgba(123, 97, 255, .35);
    background: linear-gradient(90deg, rgba(123, 97, 255, .38),
rgba(255, 79, 216, .28));
}

.btn.primary:hover {
    border-color: rgba(255, 255, 255, .18);
    background: linear-gradient(90deg, rgba(127, 231, 255, .22),
rgba(123, 97, 255, .45), rgba(255, 79, 216, .34));
}
```

```css
.btn.ghost {
    background: transparent;
    border-color: rgba(255, 255, 255, .10);
    box-shadow: none;
}

.btn.ghost:hover {
    background: rgba(255, 255, 255, .05);
    box-shadow: 0 10px 22px rgba(0, 0, 0, .18);
}

.btn.sm {
    padding: 8px 12px;
    border-radius: 11px;
    font-size: 13px;
}

.btn.lg {
    padding: 12px 16px;
    border-radius: 14px;
}

/* ========== 7) Inputs ========== */
.input,
input[type="text"],
input[type="search"],
input[type="email"],
input[type="password"],
textarea,
select {
    width: 100%;
    padding: 10px 12px;
    border-radius: 12px;
    border: 1px solid rgba(255, 255, 255, .12);
    background: rgba(255, 255, 255, .05);
    color: var(--ink-0);
    outline: none;
    box-shadow: inset 0 1px 0 rgba(255, 255, 255, .06);
}

input::placeholder,
textarea::placeholder {
    color: rgba(245, 243, 255, .45);
}

.input:focus,
input:focus,
textarea:focus,
select:focus {
    border-color: rgba(127, 231, 255, .30);
    box-shadow:
        0 0 0 3px rgba(123, 97, 255, .18),
        inset 0 1px 0 rgba(255, 255, 255, .06);
}

/* ========== 8) Cards / Tiles (Import grid look) ========== */
.grid {
    display: grid;
    gap: var(--pad-md);
}

.grid.cols-3 {
    grid-template-columns: repeat(3, minmax(0, 1fr));
}

.grid.cols-2 {
    grid-template-columns: repeat(2, minmax(0, 1fr));
}

.tile {
    border-radius: var(--radius-lg);
    padding: 18px;
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(255, 255, 255, .04);
    box-shadow: var(--shadow-sm);
    backdrop-filter: blur(var(--blur));
    -webkit-backdrop-filter: blur(var(--blur));
    min-height: 110px;
    display: flex;
    align-items: center;
    justify-content: center;
    text-align: center;
    position: relative;
    overflow: hidden;
    transition: border-color .12s ease, background .12s ease,
transform .12s ease, box-shadow .12s ease;
}

.tile:hover {
    border-color: rgba(255, 79, 216, .22);
    background: rgba(255, 255, 255, .05);
    box-shadow: 0 18px 44px rgba(0, 0, 0, .38);
    transform: translateY(-1px);
}

.tile.selected {
    border-color: rgba(255, 79, 216, .32);
    box-shadow: 0 0 0 3px rgba(255, 79, 216, .14), 0 18px 44px rgba(0,
0, 0, .42);
}
```

```
.tile::after {
    content: "";
    position: absolute;
    inset: auto -80px -90px auto;
    width: 260px;
    height: 200px;
    background: radial-gradient(circle at 30% 30%, rgba(127, 231, 255,
.16), transparent 70%);
    filter: blur(10px);
    opacity: .8;
    pointer-events: none;
}

.tile .tile-title {
    font-weight: 650;
    color: var(--ink-0);
}

.tile .tile-sub {
    font-size: 12px;
    color: var(--ink-2);
}

/* ========== 9) Tables (Import mapping look) ========== */
.table-wrap {
    border-radius: var(--radius-lg);
    overflow: hidden;
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(255, 255, 255, .03);
    backdrop-filter: blur(var(--blur));
    -webkit-backdrop-filter: blur(var(--blur));
}

table {
    width: 100%;
    border-collapse: collapse;
}

th,
td {
    text-align: left;
    padding: 12px 14px;
    border-bottom: 1px solid rgba(255, 255, 255, .08);
    color: var(--ink-1);
    vertical-align: top;
}

th {
    font-size: 12px;
```

```css
    letter-spacing: .24px;
    text-transform: uppercase;
    color: rgba(245, 243, 255, .72);
    background: rgba(123, 97, 255, .10);
}

tr:nth-child(even) td {
    background: rgba(255, 255, 255, .02);
}

tr:hover td {
    background: rgba(255, 255, 255, .04);
}

/* ========== 10) Callouts ========== */
.callout {
    border-radius: var(--radius-md);
    padding: 12px 14px;
    border: 1px solid rgba(255, 255, 255, .12);
    background: rgba(255, 255, 255, .04);
}

.callout strong {
    color: var(--ink-0);
}

.callout.info {
    border-color: rgba(123, 97, 255, .24);
    background: rgba(123, 97, 255, .10);
}

.callout.ok {
    border-color: rgba(59, 232, 195, .26);
    background: rgba(59, 232, 195, .10);
}

.callout.warn {
    border-color: rgba(255, 181, 71, .28);
    background: rgba(255, 181, 71, .10);
}

.callout.danger {
    border-color: rgba(255, 90, 106, .28);
    background: rgba(255, 90, 106, .10);
}

/* ========== 11) Document Layout (formal docs) ========== */
.page {
    width: min(var(--page-max), calc(100% - 2*var(--pad-xl)));
    margin: 0 auto;
    padding: var(--pad-xl) 0;
}

.doc {
    padding: var(--pad-xl);
    border-radius: var(--radius-lg);
}

.doc.glass {
    background: rgba(16, 10, 40, .52);
}

.doc-header {
    padding-bottom: 16px;
    margin-bottom: 18px;
    border-bottom: 1px solid rgba(255, 255, 255, .10);
    position: relative;
}

.doc-header::after {
    content: "";
    position: absolute;
    left: 0;
    right: 0;
    bottom: -1px;
    height: 1px;
    background: var(--grad-accent-soft);
    opacity: .9;
}

.doc-title {
    font-size: 30px;
    font-weight: 760;
    margin: 0 0 6px 0;
    color: var(--ink-0);
}

.doc-subtitle {
    margin: 0;
    color: var(--ink-2);
    max-width: 80ch;
}

.doc-meta {
    display: flex;
    flex-wrap: wrap;
    gap: 10px 14px;
    margin-top: 14px;
    color: var(--ink-2);
    font-size: 13px;
}

.doc-meta .field {
    display: flex;
    gap: 8px;
    align-items: baseline;
    padding: 6px 10px;
    border-radius: 12px;
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(255, 255, 255, .03);
}

.doc-meta .label {
    text-transform: uppercase;
    letter-spacing: .22px;
    font-size: 11px;
    color: rgba(245, 243, 255, .60);
}

.section {
    margin-top: 18px;
    padding: 16px 16px;
    border-radius: var(--radius-lg);
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(255, 255, 255, .03);
}

.section-title {
    margin: 0 0 10px 0;
    font-size: 18px;
    font-weight: 720;
}

.section-title .n {
    color: rgba(127, 231, 255, .85);
    margin-right: 8px;
}

ul,
ol {
    margin: 10px 0 0 20px;
    padding: 0;
}

li {
    margin: 6px 0;
    color: var(--ink-1);
}
```

```css
blockquote {
    margin: 12px 0 0;
    padding: 12px 14px;
    border-left: 3px solid rgba(123, 97, 255, .45);
    background: rgba(123, 97, 255, .08);
    border-radius: 12px;
    color: var(--ink-1);
}

pre {
    margin: 12px 0 0;
    padding: 12px 14px;
    border-radius: 14px;
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(0, 0, 0, .25);
    overflow: auto;
    color: rgba(245, 243, 255, .92);
}

code {
    padding: 2px 6px;
    border-radius: 10px;
    background: rgba(0, 0, 0, .22);
    border: 1px solid rgba(255, 255, 255, .08);
    color: rgba(245, 243, 255, .92);
}

.doc-footer {
    margin-top: 18px;
    padding-top: 14px;
    border-top: 1px solid rgba(255, 255, 255, .10);
    color: var(--ink-3);
    font-size: 12px;
}

/* ========== 12) Responsive ========== */
@media (max-width: 980px) {
    .app {
        grid-template-columns: 1fr;
    }

    .grid.cols-3 {
        grid-template-columns: repeat(2, minmax(0, 1fr));
    }
}

@media (max-width: 640px) {

    .grid.cols-3,
    .grid.cols-2 {
        grid-template-columns: 1fr;
    }

    .page {
        width: calc(100% - 2*var(--pad-md));
    }

    .doc {
        padding: var(--pad-lg);
    }
}

/* ========== 13) Print (docs shouldn’t print like a nightclub)
========== */
@media print {
    :root {
        color-scheme: light;
    }

    body {
        background: #FFFFFF !important;
        color: #111827 !important;
    }

    .glass,
    .doc,
    .section,
    .tile,
    .table-wrap,
    .topbar,
    .sidebar {
        background: #FFFFFF !important;
        border-color: #E5E7EB !important;
        box-shadow: none !important;
        backdrop-filter: none !important;
        -webkit-backdrop-filter: none !important;
        color: #111827 !important;
    }

    .muted,
    .small,
    .doc-subtitle,
    .doc-footer {
        color: #374151 !important;
    }

    a {
        text-decoration: underline;
        color: #111827 !important;
    }
    .btn {
        display: none !important;
    }
}

/* ========== 14) Timeline / Feed Components (matches your UI
screenshots) ========== */

.timeline {
    display: flex;
    flex-direction: column;
    gap: 12px;
}

.day-divider {
    display: flex;
    align-items: baseline;
    gap: 12px;
    margin: 6px 0 2px;
    color: var(--ink-0);
}

.day-divider .date {
    font-weight: 720;
    letter-spacing: .2px;
}

.day-divider .count {
    font-size: 12px;
    color: var(--ink-3);
}

.feed-card {
    position: relative;
    padding: 14px 14px;
    border-radius: var(--radius-lg);
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(255, 255, 255, .03);
    box-shadow: var(--shadow-sm);
    backdrop-filter: blur(var(--blur));
    -webkit-backdrop-filter: blur(var(--blur));
    overflow: hidden;
    display: flex;
    gap: 12px;
    align-items: flex-start;
    min-width: 0;
}

.feed-card:hover {
```

```css
    border-color: rgba(210, 185, 255, .18);
    background: rgba(255, 255, 255, .04);
}

.feed-card::after {
    content: "";
    position: absolute;
    inset: auto -120px -120px auto;
    width: 320px;
    height: 260px;
    background: radial-gradient(circle at 30% 30%, rgba(255, 79, 216,
.14), transparent 70%);
    filter: blur(10px);
    opacity: .7;
    pointer-events: none;
}

.source-badge {
    flex: 0 0 auto;
    width: 38px;
    height: 38px;
    border-radius: 14px;
    border: 1px solid rgba(255, 255, 255, .12);
    background: rgba(255, 255, 255, .05);
    box-shadow: 0 10px 22px rgba(0, 0, 0, .22);
    display: grid;
    place-items: center;
}

.feed-body {
    min-width: 0;
    flex: 1;
}

.feed-top {
    display: flex;
    align-items: center;
    justify-content: space-between;
    gap: 12px;
    margin-bottom: 6px;
}

.feed-title {
    font-weight: 700;
    color: var(--ink-0);
    font-size: 14px;
}

.feed-time {
    font-size: 12px;
    color: var(--ink-3);
    white-space: nowrap;
}

.feed-text {
    color: var(--ink-1);
    font-size: 14px;
    overflow: hidden;
    text-overflow: ellipsis;
    display: -webkit-box;
    -webkit-line-clamp: 3;
    -webkit-box-orient: vertical;
}

/* little “chip row” like your provenance pills */
.pills {
    display: flex;
    flex-wrap: wrap;
    gap: 8px;
    margin-top: 10px;
}

.pill {
    display: inline-flex;
    align-items: center;
    gap: 8px;
    padding: 6px 10px;
    border-radius: 999px;
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(255, 255, 255, .03);
    color: var(--ink-2);
    font-size: 12px;
}

.pill .dot {
    width: 8px;
    height: 8px;
    border-radius: 999px;
    background: rgba(255, 255, 255, .22);
}

.pill.accent {
    border-color: rgba(123, 97, 255, .25);
    background: rgba(123, 97, 255, .10);
    color: rgba(245, 243, 255, .86);
}

.pill.accent .dot {
    background: rgba(127, 231, 255, .70);
}
```

```css
.pill.magenta .dot {
    background: rgba(255, 79, 216, .72);
}

.pill.cyan .dot {
    background: rgba(127, 231, 255, .72);
}

/* ========== 15) Two-Panel Layout (Timeline + Right rail like
Snapshots/Provenance) ========== */

.split {
    display: grid;
    grid-template-columns: 1fr 360px;
    gap: var(--pad-lg);
    min-width: 0;
}

.rail {
    display: flex;
    flex-direction: column;
    gap: var(--pad-md);
    min-width: 0;
}

.panel {
    border-radius: var(--radius-lg);
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(255, 255, 255, .03);
    box-shadow: var(--shadow-sm);
    backdrop-filter: blur(var(--blur));
    -webkit-backdrop-filter: blur(var(--blur));
    padding: 14px 14px;
    overflow: hidden;
}

.panel-title {
    display: flex;
    align-items: center;
    justify-content: space-between;
    gap: 12px;
    margin-bottom: 12px;
}

.panel-title h2 {
    font-size: 16px;
    font-weight: 740;
}
```

```css
/* Snapshot preview cards (right rail) */
.snapshot-card {
    border-radius: 16px;
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(255, 255, 255, .03);
    overflow: hidden;
}

.snapshot-card .thumb {
    height: 120px;
    background:
        linear-gradient(180deg, rgba(255, 255, 255, .07), rgba(255,
255, 255, .02)),
        radial-gradient(360px 180px at 75% 25%, rgba(255, 79, 216,
.18), transparent 60%),
        radial-gradient(340px 170px at 15% 70%, rgba(127, 231, 255,
.12), transparent 60%);
    border-bottom: 1px solid rgba(255, 255, 255, .08);
}

.snapshot-card .meta {
    padding: 10px 12px;
    color: var(--ink-2);
    font-size: 12px;
}

/* ========== 16) Progress + Loading (calm, premium) ========== */

.progress {
    width: 100%;
    height: 6px;
    border-radius: 999px;
    background: rgba(255, 255, 255, .06);
    border: 1px solid rgba(255, 255, 255, .10);
    overflow: hidden;
}

.progress>span {
    display: block;
    height: 100%;
    width: var(--progress, 40%);
    border-radius: 999px;
    background: var(--grad-accent);
    box-shadow: 0 0 0 2px rgba(0, 0, 0, .15) inset, 0 10px 26px
rgba(123, 97, 255, .18);
}

.loading-dots {
    display: inline-flex;
    gap: 6px;
    align-items: center;
}

.loading-dots span {
    width: 6px;
    height: 6px;
    border-radius: 999px;
    background: rgba(245, 243, 255, .38);
    animation: pulse 1.25s infinite ease-in-out;
}

.loading-dots span:nth-child(2) {
    animation-delay: .15s;
}

.loading-dots span:nth-child(3) {
    animation-delay: .30s;
}

@keyframes pulse {

    0%,
    100% {
        transform: translateY(0);
        opacity: .45;
    }

    50% {
        transform: translateY(-2px);
        opacity: .95;
    }
}

/* ========== 17) Modal / Dialog (for imports, mapping, confirmations)
========== */

.backdrop {
    position: fixed;
    inset: 0;
    background: rgba(2, 1, 12, .65);
    backdrop-filter: blur(10px);
    -webkit-backdrop-filter: blur(10px);
    display: grid;
    place-items: center;
    padding: 20px;
    z-index: 50;
}
```

```css
.modal {
    width: min(720px, 100%);
    border-radius: 20px;
    border: 1px solid rgba(255, 255, 255, .12);
    background: rgba(16, 10, 40, .70);
    box-shadow: var(--shadow-lg);
    backdrop-filter: blur(22px);
    -webkit-backdrop-filter: blur(22px);
    overflow: hidden;
}

.modal-header {
    padding: 16px 18px;
    border-bottom: 1px solid rgba(255, 255, 255, .10);
    display: flex;
    align-items: center;
    justify-content: space-between;
    gap: 12px;
}

.modal-title {
    font-weight: 760;
    color: var(--ink-0);
}

.modal-body {
    padding: 16px 18px;
}

.modal-footer {
    padding: 14px 18px;
    border-top: 1px solid rgba(255, 255, 255, .10);
    display: flex;
    justify-content: flex-end;
    gap: 10px;
}

/* ========== 18) Subtle “Cosmic Dust” Background Layer (optional
utility) ========== */

.cosmic {
    position: relative;
    overflow: hidden;
}

.cosmic::before {
    content: "";
    position: absolute;
    inset: -120px;
    background:
        radial-gradient(circle at 20% 30%, rgba(255, 255, 255, .08) 0
1px, transparent 2px) 0 0/ 90px 90px,
        radial-gradient(circle at 70% 60%, rgba(255, 255, 255, .06) 0
1px, transparent 2px) 0 0/ 120px 120px;
    opacity: .35;
    filter: blur(.2px);
    pointer-events: none;
}

.cosmic::after {
    content: "";
    position: absolute;
    inset: -140px;
    background:
        radial-gradient(900px 420px at 75% 35%, rgba(255, 79, 216,
.12), transparent 65%),
        radial-gradient(820px 520px at 15% 70%, rgba(127, 231, 255,
.08), transparent 65%);
    opacity: .7;
    pointer-events: none;
}

/* ========== 19) Responsive for Split ========== */
@media (max-width: 980px) {
    .split {
        grid-template-columns: 1fr;
    }

    .rail {
        order: 2;
    }
}

/* ========== 20) Icons / Avatars / Badges ========== */

.avatar {
    width: 34px;
    height: 34px;
    border-radius: 999px;
    border: 1px solid rgba(255, 255, 255, .14);
    background:
        radial-gradient(circle at 30% 30%, rgba(127, 231, 255, .30),
transparent 60%),
        radial-gradient(circle at 70% 40%, rgba(255, 79, 216, .22),
transparent 62%),
        rgba(255, 255, 255, .04);
    box-shadow: 0 12px 26px rgba(0, 0, 0, .25);
    display: grid;
    place-items: center;
    color: var(--ink-0);
```

```css
    font-weight: 750;
    letter-spacing: .4px;
    font-size: 12px;
}

.badge {
    display: inline-flex;
    align-items: center;
    gap: 8px;
    padding: 6px 10px;
    border-radius: 999px;
    border: 1px solid rgba(255, 255, 255, .12);
    background: rgba(255, 255, 255, .04);
    color: var(--ink-1);
    font-size: 12px;
}

.badge .dot {
    width: 8px;
    height: 8px;
    border-radius: 999px;
    background: rgba(255, 255, 255, .22);
}

.badge.accent {
    border-color: rgba(123, 97, 255, .25);
    background: rgba(123, 97, 255, .12);
}

.badge.accent .dot {
    background: rgba(127, 231, 255, .75);
}

.badge.ok {
    border-color: rgba(59, 232, 195, .25);
    background: rgba(59, 232, 195, .10);
}

.badge.ok .dot {
    background: rgba(59, 232, 195, .75);
}

.badge.warn {
    border-color: rgba(255, 181, 71, .26);
    background: rgba(255, 181, 71, .10);
}

.badge.warn .dot {
    background: rgba(255, 181, 71, .78);
}
```

```css
.badge.danger {
    border-color: rgba(255, 90, 106, .26);
    background: rgba(255, 90, 106, .10);
}

.badge.danger .dot {
    background: rgba(255, 90, 106, .78);
}

/* Tiny icon button (top-right controls) */
.icon-btn {
    width: 34px;
    height: 34px;
    border-radius: 12px;
    border: 1px solid rgba(255, 255, 255, .12);
    background: rgba(255, 255, 255, .04);
    color: rgba(245, 243, 255, .82);
    display: grid;
    place-items: center;
    cursor: pointer;
    transition: background .12s ease, border-color .12s ease,
transform .12s ease;
}

.icon-btn:hover {
    background: rgba(255, 255, 255, .06);
    border-color: rgba(210, 185, 255, .18);
}

.icon-btn:active {
    transform: translateY(1px);
}

/* ========== 21) Tabs / Segmented Controls ========== */

.tabs {
    display: flex;
    gap: 8px;
    padding: 6px;
    border-radius: 16px;
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(255, 255, 255, .03);
    backdrop-filter: blur(14px);
    -webkit-backdrop-filter: blur(14px);
    width: fit-content;
}

.tab {
    padding: 8px 12px;
    border-radius: 12px;
    border: 1px solid transparent;
    background: transparent;
    color: var(--ink-2);
    font-weight: 650;
    cursor: pointer;
}

.tab:hover {
    color: var(--ink-0);
    background: rgba(255, 255, 255, .04);
    border-color: rgba(255, 255, 255, .08);
}

.tab.active {
    color: var(--ink-0);
    background: rgba(123, 97, 255, .16);
    border-color: rgba(123, 97, 255, .26);
    box-shadow: 0 10px 22px rgba(123, 97, 255, .12);
}

/* ========== 22) “Inspector” / Right Details Panel (Provenance /
Related) ========== */

.inspector {
    display: flex;
    flex-direction: column;
    gap: 12px;
}

.kv {
    display: grid;
    grid-template-columns: 1fr;
    gap: 8px;
}

.kv-row {
    display: flex;
    align-items: center;
    justify-content: space-between;
    gap: 12px;
    padding: 10px 12px;
    border-radius: 14px;
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(255, 255, 255, .03);
}

.kv-row .k {
    color: var(--ink-2);
    font-size: 13px;
```

```
}

.kv-row .v {
    color: var(--ink-0);
    font-weight: 650;
    font-size: 13px;
}

/* Related cards (like “janedoe123” preview) */
.related-card {
    display: flex;
    gap: 12px;
    align-items: flex-start;
    padding: 12px 12px;
    border-radius: 16px;
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(255, 255, 255, .03);
}

.related-card .who {
    font-weight: 720;
    color: var(--ink-0);
    font-size: 13px;
}

.related-card .what {
    color: var(--ink-2);
    font-size: 12px;
    margin-top: 2px;
}

/* ========== 23) Checklist / Mapping UI (like import mapping)
========== */

.checklist {
    display: flex;
    flex-direction: column;
    gap: 10px;
}

.check {
    display: flex;
    align-items: center;
    gap: 10px;
    padding: 10px 12px;
    border-radius: 14px;
    border: 1px solid rgba(255, 255, 255, .10);
    background: rgba(255, 255, 255, .03);
}
```

```css
.check input[type="checkbox"] {
    width: 16px;
    height: 16px;
    accent-color: var(--violet);
}

.check .label {
    color: var(--ink-1);
    font-weight: 650;
    font-size: 13px;
}

.check .hint {
    margin-left: auto;
    color: var(--ink-3);
    font-size: 12px;
}

/* Status banner “Parsing in progress…” */
.banner {
    padding: 12px 14px;
    border-radius: 16px;
    border: 1px solid rgba(255, 255, 255, .12);
    background: rgba(255, 255, 255, .04);
    display: flex;
    align-items: center;
    gap: 10px;
}

.banner .icon {
    width: 18px;
    height: 18px;
    border-radius: 8px;
    background: rgba(255, 255, 255, .06);
    border: 1px solid rgba(255, 255, 255, .10);
}

.banner .text {
    color: var(--ink-1);
    font-weight: 650;
}

.banner.progressing {
    border-color: rgba(255, 79, 216, .18);
    background: rgba(255, 79, 216, .08);
}

.banner.progressing .icon {
    background: rgba(255, 79, 216, .18);
    border-color: rgba(255, 79, 216, .22);
}

/* ========== 24) Toasts / Notifications ========== */

.toast-wrap {
    position: fixed;
    right: 18px;
    bottom: 18px;
    display: flex;
    flex-direction: column;
    gap: 10px;
    z-index: 60;
}

.toast {
    width: min(380px, calc(100vw - 36px));
    padding: 12px 14px;
    border-radius: 16px;
    border: 1px solid rgba(255, 255, 255, .12);
    background: rgba(16, 10, 40, .75);
    box-shadow: var(--shadow-lg);
    backdrop-filter: blur(18px);
    -webkit-backdrop-filter: blur(18px);
}

.toast .title {
    font-weight: 760;
    color: var(--ink-0);
    margin-bottom: 2px;
}

.toast .msg {
    color: var(--ink-2);
    font-size: 13px;
}

.toast.ok {
    border-color: rgba(59, 232, 195, .26);
}

.toast.warn {
    border-color: rgba(255, 181, 71, .26);
}

.toast.danger {
    border-color: rgba(255, 90, 106, .26);
}

/* ========== 25) Accessibility Tweaks (readability without ruining
```

```
the vibe) ========== */

:focus-visible {
    outline: 2px solid rgba(127, 231, 255, .55);
    outline-offset: 3px;
    border-radius: 12px;
}

@media (prefers-reduced-motion: reduce) {
    * {
        animation-duration: .01ms !important;
        animation-iteration-count: 1 !important;
        transition-duration: .01ms !important;
        scroll-behavior: auto !important;
    }
}

/* ========== 26) Optional “Hero” Title Treatment (matches big center
titles) ========== */

.hero {
    text-align: center;
    padding: 28px 18px;
}

.hero h1 {
    font-size: 34px;
    font-weight: 780;
    margin: 0 0 10px;
}

.hero .lede {
    margin: 0 auto;
    max-width: 74ch;
    color: rgba(245, 243, 255, .70);
}

.hero .rule {
    width: 220px;
    height: 2px;
    margin: 18px auto 0;
    border-radius: 999px;
    background: var(--grad-accent-soft);
}

/* ========== 27) Final responsive touches ========== */
@media (max-width: 860px) {
    .app {
        padding: var(--pad-lg);
        gap: var(--pad-md);
    }

    .topbar {
        flex-wrap: wrap;
    }

    .search {
        max-width: 100%;
    }
}
```

FILE: docs/02_DESIGN/DESIGN_SYSTEM/TOKENS.md

Design Tokens

This document serves as the master dictionary for the **Nebula** v2.0 design tokens. These tokens are implemented as CSS Custom Properties (`:root`) in `theme.css`.

1. Typography

Token	CSS Variable	Value Reference
Sans	`--font-sans`	Inter, system-ui
Mono	`--font-mono`	JetBrains Mono, monospace
Size XS	`--text-xs`	12-14px (Fluid)
Size SM	`--text-sm`	14-16px (Fluid)
Size Base	`--text-base`	16-18px (Fluid)
Size LG	`--text-lg`	18-20px (Fluid)
Size XL	`--text-xl`	20-24px (Fluid)
Size 2XL	`--text-2xl`	24-32px (Fluid)
Size 3XL	`--text-3xl`	30-40px (Fluid)
Size 4XL	`--text-4xl`	36-48px (Fluid)

2. Color Semantic Roles

See `COLOR_SYSTEM.md` for detailed palettes.

Token	Role
`--ink-0`	High Emphasis Text (Headings)
`--ink-1`	Medium Emphasis Text (Body)
`--ink-2`	Low Emphasis Text (Muted)
`--ink-3`	Decorative Text (Subtle)

Token	Role
`--panel`	Base Layer (Sidebar, App Background)
`--panel-2`	Surface Layer (Cards)
`--panel-3`	Top Layer (Modals, Popovers)
`--stroke`	Border Color
`--hairline`	Subtle Inner Border

3. Brand Accents

Token	Color	Usage
`--violet`	Violet	Primary Action
`--cyan`	Cyan	Information, Focus
`--magenta`	Magenta	Creative, AI
`--ok`	Mint Green	Success
`--warn`	Amber	Warning
`--danger`	Red	Error/Destructive

4. Geometry & Effects

Radii

- `--radius-sm`: 10px (Inner elements)
- `--radius-md`: 14px (Buttons, Inputs)
- `--radius-lg`: 20px (Cards, Modals)

Blur

- `--blur`: 24px (Standard backdrop blur)

Shadows

- `--shadow-lg`: Deep elevation shadow + inset highlight.
- `--shadow-glow`: Colored diffusion.

Animation Easings

- `--ease-out`: General transitions.
- `--ease-squish`: Button interactions.
- `--ease-elastic`: Bouncy reveals.

5. Textures & Gradients

- `--noise`: SVG Turbulence filter for texture.
- `--grad-accent`: Cyan → Violet → Magenta.
- `--grad-sheen`: Subtle glass reflection.

FILE: docs/02_DESIGN/DESIGN_SYSTEM/TYPOGRAPHY.md

Typography System

- **Headings**: `Outfit` (Geometric Sans) - Medium weight for H1-H3.
- **Body**: `Inter` (Standard Sans) - Regular for general copy.
- **Monospace**: `JetBrains Mono` - Used for Citation IDs and technical metadata.

Sizing (Modular Scale 1.25)

- **H1**: 48px / 3rem
- **H2**: 36px / 2.25rem
- **H3**: 24px / 1.5rem
- **Body**: 16px / 1rem
- **Caption**: 12px / 0.75rem

FILE: docs/02_DESIGN/IA/INFORMATION_ARCHITECTURE.md

Information Architecture — Memoir.ai

The structural organization of the Memoir.ai local ecosystem.

1. Global Hierarchy

- **Library (Vault)**: The top-level container for all data and configuration.
  - **Timeline**: The chronological index of all `Events`.
  - **People**: The resolved entity graph of `Participants`.
  - **Drafts**: The library of `Narratives` and `Snapshots`.
  - **Sources**: The collection of `DataSources` and `ImportJobs`.

2. Logical Grouping

- **Core Experience**: Timeline, Search, Event View.
- **Intelligence Layer**: Generation, Verification, Relationship Analytics.
- **Data Management**: Ingestion, Validation, Storage, Export.
- **Commercials**: Subscriptions, Usage Metering, Billing.

3. IA Visual Map (High Level)

```
graph TD
    A[App Shell] --> B[Timeline]
    A --> C[Snapshots]
    A --> D[Imports]
    A --> E[Settings]
    A --> F[Billing]
    B --> G[Event Details]
    B --> H[Search Results]
    C --> I[Editor]
    C --> J[Archives]
    D --> K[Source Manager]
    D --> L[Import Wizard]
```

FILE: docs/02_DESIGN/IA/NAV_MODEL.md

Navigation Model — Memoir.ai

Defines how users move through the Memoir.ai application.

1. Primary Sidebar Navigation

- **Top 5 Priority Tabs**:
    1. **Home**: Dashboard overviews.
    2. **Timeline**: The master feed.
    3. **Snapshots**: Narrative creation.
    4. **Imports**: Data ingestion.
    5. **Settings**: Configuration.

2. Modal & Contextual Flows

- **Modal Overlays**: Used for destructive actions (Delete Vault), complex configuration (Import Wizard), and deep event inspection.
- **Inspectors**: Right-side drawers for metadata without losing timeline context.

3. Back-Navigation Strategy

- The app maintains a session-level history stack.
- Deep links (e.g., clicking a citation in a snapshot) push the timeline view to the stack, allowing a "Back" button to return the user to their writing.

4. Command Palette (`Cmd+K`)

- Quick access to all primary routes.
- Recent search history and specific friend profiles.

FILE: docs/02_DESIGN/IA/ROUTES_MAP.md

Routes Map — Memoir.ai

Internal application routing structure (React Router / Electron IPC).

1. Core Endpoints

- `/app/home`: Dashboard & Widgets.
- `/app/timeline`: Master Feed.
- `/app/timeline/event/:id`: Deep link to specific memory.
- `/app/snapshots`: Library of narratives.
- `/app/snapshots/new`: Wizard for generation.
- `/app/snapshots/edit/:id`: Detailed editor.
- `/app/imports`: Active & Past jobs.
- `/app/imports/wizard`: Multi-step ingestion flow.
- `/app/billing`: Plans & Payments.
- `/app/settings/:tab`: Tabbed config (vault, privacy, etc.).

2. Protected Routes

- All `/app/*` routes require an unlocked vault status.
- `/auth/login`: The lock screen.
- `/auth/onboarding`: Initial setup flow.

3. Deep Linking Parameters

- `?t=timestamp`: Scroll to specific time.
- `?q=query`: Pre-filled search state.
- `?source=id`: Filter results by specific source.

FILE: docs/02_DESIGN/UX_SPECS/SCREEN_INDEX.md

Screen Spec Index

Mapping all application states to their technical documentation.

Screen	Spec Path	Status
App Shell	APP_SHELL.md	COMPLETE
Auth	AUTH_LOGIN.md	COMPLETE
Timeline	TIMELINE.md	COMPLETE
Snapshots	SNAPSHOTS.md	COMPLETE
Billing	BILLING.md	COMPLETE

FILE: docs/02_DESIGN/UX_SPECS/SCREEN_SPECS/APP_SHELL.md

Screen Spec: App Shell

The App Shell provides the persistent layout and navigation framework for Memoir.ai.

1. Layout Zones

- **Sidebar (Left)**: Primary navigation links (Timeline, Snapshots, Imports, Settings, Billing). Includes the vault status indicator.
- **Topbar**: Persistent search bar, active job progress indicator (Job Runner), and user profile/vault selector.
- **Main Content**: Dynamic area for screen-specific content.
- **Status Bar (Bottom)**: Local database health, last sync time, and "Offline-first" confirmation.

2. Interactive Components

- **Nav Links**: Active state highlighted with "Nebula" Glow (Cyan/Violet gradient).
- **Search Bar**: Expands on focus. Supports command palette triggers (`Cmd+K`).
- **Sync Badge**: Pulses during active background parsing.

3. Transitions

- Main content area uses a soft 200ms opacity fade between routes.
- Sidebar collapses/expands with a smooth width transition.

4. States

- **Locked**: Main content is blurred; persistent overlay for vault passphrase entry.
- **Processing**: Global progress bar appears at the top of the content area.

FILE: docs/02_DESIGN/UX_SPECS/SCREEN_SPECS/AUTH_LOGIN.md

Screen Spec: Auth Login

The primary gateway for unlocking an existing local vault.

1. Visual Design

- **Background**: Dark Nebula gradient (Violet to Deep Space Black).
- **Center Card**: Glassmorphic container with 20% opacity and subtle border glow.
- **Identity**: Large Memoir.ai logo with "Cinematic Precision" tagline.

2. Components

- **Vault Path Display**: Shows the location of the detected `.sqlite` file.
- **Passphrase Input**: Secure text entry with a "Nebula" focus ring.
- **Unlock Button**: Primary action. Glows stronger on hover.
- **"Forgot Passphrase" link**: Directs user to the "Lost Key" warning document.

3. Interaction

- On successful unlock, the card slides upward and fades out.
- On failure, a subtle vibration animation and a "Nebula Red" error message appear.

4. Error States

- **Invalid Key**: "Incorrect passphrase. Please try again or restore from backup."
- **Vault Missing**: "No vault found at this location. Connect or create one."

FILE: docs/02_DESIGN/UX_SPECS/SCREEN_SPECS/AUTH_SIGNUP.md

Screen Spec: Auth Signup (Vault Creation)

The onboarding flow for creating a new local-first library.

1. Visual Design

- **Progress Stepper**: Horizontal dots at the top (Philosophy -> Location -> Security -> Finish).
- **Nebula Accents**: Dynamic light beams that guide the eye to the primary action.

2. Components

- **Directory Picker**: Native OS folder selection button.
- **Passphrase Creation**:
  - Input field with visibility toggle.
  - Strength meter (Weak/Moderate/Strong/Cinematic).
  - Confirmation field.
- **"Start Journey" Button**: Disabled until passphrase matches and meets "Strong" criteria.

3. User Guidance

- Inline alerts explaining that Memoir.ai cannot recover lost passwords.
- Privacy badges confirming "100% On-Device Processing".

4. States

- **Creating...**: Shimmering progress bar while folders and DB files are initialized.

- **Success**: Confetti-like Nebula particles and immediate transition to the Import Wizard.

FILE: docs/02_DESIGN/UX_SPECS/SCREEN_SPECS/BILLING.md

Screen Spec: Billing

Manage subscriptions, payment methods, and usage limits with premium cinematic clarity.

1. Primary Components

- **Plan Summary Card**: A glassmorphic panel showing the current active tier (Free/Pro/Cinematic).
- **Stripe Pricing Table**: Three-column view (Solo, Pro, Team - as per market spec) with feature comparisons.
- **Usage Metrics**: Radial charts showing AI snapshot tokens consumed and storage used.
- **Invoice History**: A clean list with Download PDF actions.

2. Interaction Flows

- **Upgrade**: Clicking "Upgrade" opens the Stripe Checkout in a secure modal/redirect.
- **Cancellation**: A multi-step retention flow ("Why are you leaving?") with a clear confirmation action.

3. Visual States

- **Trial Active**: A persistent banner at the bottom showing "! 5 days remaining in your pro trial".
- **Payment Failed**: Red glow intensity increases on the card; "Update Payment Method" button pulses.

4. States

- **Loading**: Skeleton loaders for the pricing cards (shimmering violet).
- **Empty**: Default to Free Tier layout if no subscription is detected.

FILE: docs/02_DESIGN/UX_SPECS/SCREEN_SPECS/IMPORTS.md

Screen Spec: Imports (Wizard)

The high-stakes flow for ingesting archives into the vault.

1. Components

- **Source Grid**: Interactive cards for iMessage, WhatsApp, Instagram, etc.

- **Path Validator**: A text input with a real-time check mark (Green for valid, Red for invalid path).
- **Import Queue**: A list of pending/active/completed jobs with progress percentages.
- **Log Viewer**: An expandable panel showing real-time parser output.

2. Interaction

- **Drag & Drop**: Users can drop `.db` or `.zip` files directly onto the source cards.
- **Deep Diagnostics**: Clicking a failed job reveals the specific line/record that caused the error.

3. Visual Feedback

- **Job Pulse**: Active jobs have a circular progress bar with a secondary "Pulse" animation to show compute activity.
- **Success Confetti**: Subtle violet particles when an archive is fully indexed.

4. States

- **Validating**: Small spinner next to the file path.
- **Partial Success**: Orange warning icon; "98% Ingested - 2 records skipped".

FILE: docs/02_DESIGN/UX_SPECS/SCREEN_SPECS/LIBRARY_HOME.md

Screen Spec: Library Home

The high-level dashboard for your digital history health.

1. Dashboard Widgets

- **Total Memories Count**: Animated counter for every unique normalized event.
- **Source Health**: Grid showing connectivity and sync status for all imported channels.
- **Recent Snapshots**: A carousel of the latest 3 AI-generated narratives.
- **Relationship Radar**: A visualization of the most frequent communication partners.

2. Layout

- Split view: Statistics on the left, "Quick Actions" (Import, Build Snapshot, Search) on the right.

3. Navigation

- One-click access to the full **Unified Timeline** or specific **Source Settings**.

4. States

- **Fresh Vault**: "Your history is empty. Start your first import." with a large central CTA.
- **Syncing**: Widgets have a "Back-Syncing" overlay to show that data is currently being updated.

FILE: docs/02_DESIGN/UX_SPECS/SCREEN_SPECS/SETTINGS.md

Screen Spec: Settings

Comprehensive configuration for the local backend and UI.

1. Tabbed Navigation

- **General**: Theme (Dark/Light/Nebula), Language, Startup behavior.
- **Vault**: Path management, Re-encryption/Passphrase change, Data Wipe.
- **Privacy**: Local analytics toggle, Sensitive content filters, AI guardrail intensity.
- **Advanced**: Index rebuilding, Log levels, Storage optimization.

2. Components

- **Switch Toggles**: Cinematic glow on active state.
- **Destructive Buttons**: "Wipe Vault" is highlighted in a distinct "Warning Red" with a confirmation delay.
- **Path Display**: Non-editable field with a "Browse" button.

3. Feedback

- Immediate "Saved" toast notification for non-critical changes.
- Modal confirmation for changes requiring a vault restart.

4. States

- **Repairing**: Settings are disabled; progress bar shows index rebuild status.
- **Disconnected**: Warning banner if the vault path is no longer available.

FILE: docs/02_DESIGN/UX_SPECS/SCREEN_SPECS/SNAPSHOTS.md

Screen Spec: Snapshots (Drafting & Archives)

The creative suite for managing on-device AI narratives.

1. Components

- **Draft Editor**: A rich-text editor with "Nebula" inspired glassmorphic toolbar.
- **Citation Sidebar**: A permanent panel on the right that displays the source evidence corresponding to the highlighted text.
- **Version History**: A vertical list of previously generated drafts with "Revert" and "Compare" actions.
- **Generation Controls**: Tone slider (Clinical <-> Lyrical), Length selector, and "Regenerate" button.

2. Interaction

- **Citation Hover**: Hovering over a citation bubble in the text highlights the linked event in the sidebar.
- **Live Verification**: As the user edits, the system performs real-time checks to see if claims still align with cited evidence.

3. Visual Style

- **Focus Mode**: Collapses the sidebar and topbar to provide a cinematic, distraction-free writing experience.
- **Unverified Tags**: Sentences lacking citations are marked with a subtle "Nebula Red" dotted underline.

4. States

- **Generating**: A central "Brain" icon pulses with violet light beams while the local model runs.
- **Empty**: "No narratives yet. Choose a time range from the timeline to begin."

FILE: docs/02_DESIGN/UX_SPECS/SCREEN_SPECS/TIMELINE.md

Screen Spec: Timeline (Master Feed)

The core analytical view of the unified digital history.

1. Primary Components

- **Infinite Feed**: A vertical stream of events with "Smart Date" headers.
- **Event Cards**: Condensed views for messages, photos, and posts. Includes source icons and participant avatars.
- **Timeline Ruler**: A right-side vertical axis allowing users to scrub through years/months rapidly.
- **Filter Bar**: Facet selection (People, Source, Type, Date Range).

2. Conversation Viewer (Modal)

- Clicking a message expands a thread view.
- **Thread Linkage**: Reconstructs replies even if they occurred across multiple sources (e.g., iMessage and WhatsApp merge).

3. Visual Polish

- **Event Clustering**: Consecutive messages from the same person are grouped visually to reduce clutter.
- **Media Previews**: High-quality thumbnails with a cinematic blur-in effect on scroll.

4. States

- **No Results**: A "Galaxy" themed empty state with a "Clear Filters" button.
- **Deep Loading**: Shimmer effect on cards when fetching large blocks from the encrypted database.

FILE: docs/02_DESIGN/A11Y/CONTRAST_RULES.md

Contrast Rules: Nebula Theme

Ensuring readability across the cinematic Violet/Cyan/Magenta palette.

1. Core Backgrounds

- **Primary Background**: `#0A0118` (Deep Space Violet).
- **Glass Panel**: `rgba(255, 255, 255, 0.05)` with `backdrop-filter: blur(12px)`.

2. Text vs. Background Ratios (WCAG AA)

Element Type	Hex Code	Background Hex	Ratio	Status
Primary Body	`#F1F1F1`	`#0A0118`	15.8:1	PASS
Secondary/ Caption	`#B0A6C1`	`#0A0118`	8.2:1	PASS
Cyan Accent	`#00F2FF`	`#0A0118`	9.5:1	PASS
Magenta Highlight	`#FF00D4`	`#0A0118`	4.8:1	PASS

3. Interaction Contrasts

- **Active Nav Icons**: Must use Cyan (`#00F2FF`) to ensure distinction from inactive state.
- **Error States**: Use a vibrant Red (`#FF3B30`) with a glow effect, ensuring ≥ 4.5:1 against the dark background.

- **Focus Rings**: Standardised to 2px solid Cyan with 4px outer violet glow.

4. Visualization Rules

- Graphs and charts must use contrasting textures or patterns in addition to color to distinguish data series.

FILE: docs/02_DESIGN/A11Y/KEYBOARD_NAV_SPEC.md

Keyboard Navigation Specification

Comprehensive physical interaction mappings for Memoir.ai.

1. Global Shortcuts

- `Cmd+K` / `Ctrl+K`: Open Command Palette / Global Search.
- `Cmd+,`: Open Settings.
- `Esc`: Close any active Modal or Inspector.
- `Alt+T`: Jump to Timeline.
- `Alt+S`: Jump to Snapshots.

2. Timeline Interaction

- `J`: Scroll down one event.
- `K`: Scroll up one event.
- `Enter`: Open Conversation Viewer for selected message.
- `Space`: Preview media element.

3. Snapshot Editor

- `Cmd+S`: Save local draft.
- `Cmd+R`: Trigger "Regenerate" job for selected paragraph.
- `Tab`: Cycle between Editor and Citation Sidebar.

4. Focus Standards

- **Trap Focus**: Modals (like the Import Wizard) MUST trap keyboard focus until closed or canceled.
- **Semantic Order**: Navigation follows the physical layout (Topbar -> Sidebar -> Content -> Status).

FILE: docs/02_DESIGN/A11Y/WCAG_CHECKLIST.md

WCAG 2.1 Checklist (Memoir.ai)

Internal audit tool for ensuring AA compliance on-device.

1. Perceivable

- [ ] **Text Alternatives**: All media (photos, icons) have descriptive `alt` text.
- [ ] **Adaptable**: Vault status and job progress are detectable by screen readers.
- [ ] **Distinguishable**: "Nebula" color contrast meets AA targets (verified in CONTRAST_RULES.md).

2. Operable

- [ ] **Keyboard Accessible**: All functionality available via keys (verified in KEYBOARD_NAV_SPEC.md).
- [ ] **Enough Time**: Passphrase entry and wizards have no time-outs that destroy progress.
- [ ] **Navigation**: Clear page titles and focus indicators.

3. Understandable

- [ ] **Readable**: Language set correctly in HTML lang attribute.
- [ ] **Predictable**: Navigation sidebar remains in a fixed position.
- [ ] **Input Assistance**: Form errors (in Settings/Auth) are clearly labeled with suggestions for correction.

4. Robust

- [ ] **Parsing**: HTML tags are used semantically and correctly closed.
- [ ] **Compatible**: UI components use standard ARIA roles for compatibility with VoiceOver/NVDA.

Memoir.ai — AI Subsystem Technical Manual

Architecture, Execution, and Operational Reference

Document Version: 1.0

Status: Engineering Reference

Owner: AI Systems Engineering

Last Updated: YYYY-MM-DD

----

1. Purpose

This technical manual defines the architecture, operational flow, safeguards, and integration requirements of the Memoir.ai AI subsystem. It serves as the authoritative reference for engineers implementing, maintaining, and extending AI-powered features within the platform.

----

2. Scope

This manual covers:

• AI subsystem architecture

• Snapshot generation pipeline

• Evidence extraction and context construction

• Model inference and orchestration

• Citation anchoring logic

• Hallucination prevention mechanisms

• Narrative versioning and regeneration logic

• Evaluation and quality measurement

• Safety and privacy safeguards

• Performance considerations

• Failure recovery patterns

User-facing documentation is out of scope.

----

3. System Overview

Memoir.ai AI operations run primarily on-device to preserve privacy and prevent unintended data exposure. Processing occurs through isolated worker processes interacting with the encrypted vault.

Subsystem components include:

• Evidence ingestion pipeline

• Prompt context builder

• Local LLM inference engine

• Citation engine

• Post-processing verification layer

• Version storage system

• Evaluation and safety checks

All AI outputs must remain traceable to source evidence.

----

4. Architectural Components

4.1 Evidence Extraction Engine

Normalizes imported event data into structured records used for AI prompts. Responsibilities include:

• Participant detection

• Temporal anchor resolution

• Event clustering

• Media association

• Deduplication handling

Output feeds prompt construction.

4.2 Prompt Context Builder

Constructs context windows supplied to inference models. Context includes:

• Evidence summaries

• Timeline anchors

• Entity mappings

• Tone and length parameters

• User preferences

Context assembly prioritizes factual grounding.

4.3 Model Inference Engine

Local models generate narratives or summaries. Responsibilities include:

• Model loading and lifecycle control

• Resource scheduling

• Prompt injection

• Token limit enforcement

• Output streaming support

Models may vary by hardware capability.

4.4 Citation Engine

Maps narrative claims back to event identifiers.

Rules:

• Citations must support explicit claims.

• Claims without evidence receive no citation.

• Conflicting evidence produces conflict markers.

----

5. Snapshot Generation Pipeline

Execution stages:

1. User selects event range or conversation.

2. Evidence set assembled.

3. Entities and timeline anchors extracted.
4. Prompt context constructed.
5. Local inference executed.
6. Output passed through verification checks.
7. Citations linked to events.
8. Sanitization and filtering applied.
9. Narrative stored with version metadata.

Failures trigger retries or error states.

----

6. Narrative Versioning

Version rules:

• Each regeneration creates immutable versions.

• Manual edits create protected user versions.

• Automatic regeneration cannot overwrite manual edits.

• Only five recent versions retained unless pinned.

Version history enables rollback and comparison.

----

7. Hallucination Prevention

7.1 Natural Language Inference Guard

Each generated sentence is validated against cited evidence. Contradictions trigger rejection or revision.

7.2 Entity Validation Guard

Names, locations, and dates must exist in evidence metadata. Invented entities cause hard failures.

7.3 Tone Guard

Overly assumptive or moralizing language is flagged and softened.

----

8. Evaluation Framework

Evaluation occurs during testing and optional runtime checks.

Metrics include:

• Claim accuracy rate

• Citation density

• Narrative flow quality

• Hallucination frequency

• Generation latency

Snapshots must exceed defined thresholds to be marked verified.

----

9. Safety & Privacy Constraints

AI must not:

• Provide medical, legal, or psychological diagnoses.

• Infer criminal or malicious intent.

• Provide financial advice.

• Expose personal identifiers in public outputs.

Sensitive data must be redacted when exporting summaries.

----

10. Performance Considerations

Operational constraints include:

• Snapshot generation targets under 10 seconds on modern hardware.

• Worker memory usage capped per job.

• Background jobs throttled to prevent UI blocking.

• Large evidence sets chunked for inference.

Performance degradation triggers diagnostics alerts.

----

11. Failure Modes & Recovery

Common failures include:

Inference timeout:

• Retry inference with reduced context.

Citation mismatch:

• Re-run citation mapping.

Guard failure:

• Mark snapshot unverified and prompt regeneration.

Worker crash:

• Restart worker and restore job state.

----

12. Model Lifecycle Management

Responsibilities include:

• Hardware capability detection

• Model tier selection

• Efficient model loading

• Graceful shutdown of idle models

• Memory cleanup after inference

Model upgrades must preserve output compatibility.

----

13. Integration Interfaces

Subsystem interacts with:

• Vault database for evidence retrieval

• Job runner for background tasks

• UI components for progress display

• Entitlement system for generation limits

All IPC messages require schema validation.

----

14. Operational Best Practices

Engineering teams should:

• Validate evidence normalization pipelines regularly.

• Monitor hallucination guard effectiveness.

• Benchmark inference latency across releases.

• Maintain backward compatibility of narrative formats.

• Run regression evaluation tests before releases.

----

15. Conclusion

This technical manual defines how Memoir.ai generates trustworthy narratives while preserving user privacy and data integrity. Proper adherence ensures AI outputs remain verifiable, performant, and safe while supporting future platform evolution.

Memoir.ai — API Reference Manual

Backend, Sync, and IPC Interface Specifications

Document Version: 1.0

Status: Engineering Reference

Owner: Platform API Team

Last Updated: YYYY-MM-DD

----

1. Purpose

This API Reference Manual documents all backend, synchronization, and inter-process communication interfaces used within Memoir.ai. It serves as the canonical engineering reference for implementing frontend integration, backend services, and external integrations.

----

2. Scope

This manual covers:

• Local REST API endpoints

• IPC communication interfaces

• Sync service endpoints

• Billing and entitlement endpoints

• Import and job management APIs

• Timeline and search APIs

• Snapshot generation APIs

• Export and deletion APIs

• Media APIs

• Event and webhook schemas

• Error handling formats

UI implementation details are out of scope.

----

3. Communication Model

Memoir.ai uses three primary communication paths:

1. IPC Bridge

   Renderer ↔ Backend for vault operations.

2. Local REST APIs

   Backend endpoints for metadata and operations.

3. Sync Service APIs

   Optional cloud metadata synchronization.

Sensitive vault operations must occur via IPC channels.

----

4. Authentication Model

Vault state determines API access:

Locked Vault:

• Only authentication endpoints accessible.

• Other endpoints return forbidden errors.

Unlocked Vault:

• All local endpoints available.

• Backend holds encryption keys in memory only.

Requests include a secure vault session token.

----

5. Standard API Response Format

All endpoints return structured JSON:

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

Errors populate the error object with recovery guidance.

----

6. Vault APIs

POST /api/vault/init

Initialize new vault.

POST /api/vault/unlock

Unlock encrypted vault.

POST /api/vault/lock

Lock active vault session.

POST /api/vault/change-passphrase

Change vault encryption credentials.

----

7. Timeline APIs

GET /v1/timeline

Retrieve paginated timeline events.

GET /v1/timeline/count

Total event count.

GET /v1/timeline/stats

Timeline event distribution metrics.

----

8. Search APIs

POST /v1/search/query

Execute hybrid lexical and semantic search.

GET /v1/search/history

Retrieve previous queries.

Parameters include alpha weighting and result limits.

----

9. Import & Jobs APIs

POST /v1/imports/jobs

Create ingestion job.

GET /v1/imports/jobs

List ingestion history.

GET /v1/imports/jobs/:id

Retrieve job status and logs.

DELETE /v1/imports/sources/:id

Remove import source.

----

10. Snapshot APIs

POST /v1/snapshots/generate

Generate AI narrative.

GET /v1/snapshots

List snapshots.

GET /v1/snapshots/:id

Retrieve narrative and citations.

PATCH /v1/snapshots/:id

Save edits.

POST /v1/snapshots/:id/regenerate

Create new narrative version.

----

11. Media APIs

GET /v1/media

Search media assets.

GET /v1/media/:id/url

Retrieve high-resolution media URL.

GET /v1/media/:id/thumbnail

Retrieve preview image.

----

12. Export & Delete APIs

POST /v1/export/request

Initiate vault export.

GET /v1/export/status/:id

Export job status.

GET /v1/export/download/:id

Download export bundle.

POST /v1/account/delete

Begin account deletion.

POST /v1/account/delete/confirm

Confirm deletion.

----

13. Billing APIs

GET /v1/billing/status

Retrieve subscription status.

GET /v1/billing/plans

Retrieve plan options.

POST /v1/billing/checkout

Create Stripe checkout session.

DELETE /v1/billing/cancel

Cancel subscription.

----

14. Sync APIs

POST /v1/sync/heartbeat

Update cloud sync state.

GET /v1/jobs/:id/status

Check remote job status.

----

15. Event Schemas

VAULT_LOCKED Event

Emitted when vault session locks.

IMPORT_PROGRESS Event

Provides ingestion progress updates.

SNAPSHOT_GENERATED Event

Signals narrative generation completion.

----

16. Webhooks

Stripe Events:

• checkout.session.completed

• customer.subscription.deleted

• invoice.payment_failed

Webhooks must validate signatures before processing.

----

17. Error Handling & Taxonomy

Errors follow structured codes:

AUTH_* Authentication failures

VALT_* Vault integrity issues

IMPT_* Import pipeline failures

AI_* Narrative generation issues

BILL_* Billing problems

Errors include recovery guidance for user action.

----

18. Implementation Best Practices

Engineering guidance:

• Validate inputs at all endpoints.

• Sanitize IPC messages.

• Maintain backward compatibility.

• Monitor performance metrics.

• Avoid exposing sensitive data.

----

19. Conclusion

This API reference provides a consistent contract for integrating Memoir.ai backend and sync services. Maintaining strict adherence ensures platform stability, secure communication, and predictable feature evolution.

Memoir.ai — Backend Implementation Handbook

Backend Services, Execution Logic, and Engineering Implementation Guide

Document Version: 1.0

Status: Engineering Implementation Reference

Owner: Backend Engineering Team

Last Updated: YYYY-MM-DD

----

1. Purpose

This handbook provides implementation guidance for engineers building and maintaining the Memoir.ai backend systems. It documents service responsibilities, execution flows, processing logic, and operational requirements required to implement ingestion, storage, AI processing, search, and export capabilities.

----

2. Scope

This handbook covers:

• Backend architecture structure

• Local service implementation

• Worker execution model

• Import and ingestion pipelines

• Job runner mechanics

• AI snapshot processing

• Search and indexing integration

• Storage and vault interactions

• API implementation patterns

• Failure handling and recovery

• Security and entitlement enforcement

UI implementation and frontend logic are out of scope.

----

3. Backend Architecture Overview

Memoir.ai backend runs within the Electron environment using Node.js services responsible for secure vault access and data processing.

Core layers include:

• Local REST service layer

• IPC bridge services

• Job runner and queue manager

• Import and normalization pipelines

• Narrative generation services

• Search indexing services

• Export and packaging services

• Security and encryption controls

Heavy operations run in background workers.

----

4. Service Layer Responsibilities

Backend services must:

• Provide timeline and search data

• Execute ingestion jobs

• Manage snapshot generation

• Handle vault encryption lifecycle

• Provide export operations

• Validate entitlement gating

• Handle configuration persistence

All services must be non-blocking to preserve UI responsiveness.

----

5. Vault Interaction Model

Vault operations include:

• Database connection management

• Encryption key loading and clearing

• Transaction-safe writes

• Media asset storage

• Integrity verification

Database writes must use transaction boundaries for consistency.

Keys must never be persisted to disk.

----

6. Import & Ingestion Pipeline Implementation

Pipeline stages include:

1. Job record creation
2. Raw archive parsing
3. Schema validation
4. Data normalization
5. Batch insertion
6. Indexing trigger

Pipeline must support:

• Resumable imports
• Batch checkpointing
• Failure recovery
• Duplicate detection

Import workers operate independently of UI.

----

7. Job Runner Implementation

Jobs are stored in queue tables and processed by worker threads.

Lifecycle:

1. Enqueue job
2. Worker claims task
3. Execution begins
4. Progress events emitted
5. Completion or failure persisted

Required features:

• Priority scheduling
• Memory throttling
• Retry handling
• Crash recovery

Jobs must support safe resumption after interruption.

----

8. Worker Execution Model

Workers handle:

• Imports

• Indexing

• Snapshot generation

• Media processing

Rules:

• Limit worker count to CPU cores minus one

• Pause jobs under memory pressure

• Persist progress periodically

Workers must communicate progress through IPC.

----

9. AI Snapshot Backend Flow

Snapshot generation involves:

1. Event sampling
2. Context assembly
3. Local inference execution
4. Citation mapping
5. Verification checks
6. Version persistence

Backend must validate citation mapping before storing narratives.

Manual edits must create new versions.

----

10. Search & Indexing Integration

Backend responsibilities include:

• Maintaining FTS indices

• Managing vector search stores

• Updating indices after ingestion

• Executing hybrid queries

Index builds must run in background jobs.

Search operations must remain under latency targets.

----

11. API Implementation Patterns

APIs expose vault operations to the UI.

Rules:

• Use structured JSON responses

• Include error codes and recovery hints

• Validate all inputs

• Reject requests when vault locked

Sensitive operations must occur through IPC, not REST endpoints.

----

12. Export Service Implementation

Export operations include:

• Snapshot of vault state

• Event and narrative serialization

• Media packaging

• ZIP archive creation

• Integrity checksum generation

Exports must function offline.

Export tasks run as background jobs.

----

13. Failure Handling & Recovery

Common backend failures include:

Import interruption:

• Resume from checkpoint.

Database corruption:

• Trigger integrity repair.

Worker crash:

• Restart worker and resume jobs.

Service implementations must log failures without exposing user data.

----

14. Security Enforcement Implementation

Backend must enforce:

• Vault unlock checks

• Entitlement gating

• Input sanitization

• IPC message validation

• Memory cleanup after encryption use

Renderer processes must never access filesystem directly.

----

15. Entitlement & Billing Checks

Before executing AI or import operations:

• Validate subscription tier

• Confirm usage limits

• Deny operations exceeding limits

Entitlements cached locally for offline behavior.

----

16. Logging & Diagnostics Implementation

Logging must capture:

• Job lifecycle events

• Import failures

• Service crashes

• Performance anomalies

Logs must exclude private content and identifiers.

----

17. Development Best Practices

Backend engineering guidance:

• Maintain schema backward compatibility

• Validate ingestion correctness

• Benchmark job performance

• Monitor memory usage

• Test recovery scenarios

Refactoring must preserve vault compatibility.

----

18. Deployment Considerations

Backend releases must ensure:

• Schema compatibility

• Migration success

• Worker stability

• Vault integrity

Release validation required before distribution.

----

19. Conclusion

This handbook provides backend engineers with implementation guidance ensuring Memoir.ai backend services remain reliable, performant, and privacy-preserving while supporting ingestion, search, and AI narrative generation at scale.

Memoir.ai — Compliance & Regulatory Summary Guide (Expanded)

Document Version: 1.1

Status: Enterprise Operational Reference

Owner: Platform Architecture & Operations

Last Updated: YYYY-MM-DD

Regulatory Scope

Summarizes alignment with privacy and data portability regulations.

Data Ownership

Users retain full control of vault data and export capabilities.

Deletion Support

Supports vault deletion and data export for compliance needs.

Portability Guarantees

Data export uses open formats ensuring portability.

Revision History

Expanded compliance mapping.

Memoir.ai — Compliance & Regulatory Summary Guide

Document Version: 1.0

Status: Engineering / Enterprise Reference

Last Updated: YYYY-MM-DD

Purpose

Summarize regulatory alignment for enterprise evaluators.

Scope

GDPR, CCPA, and privacy-aligned operational behaviors.

Data Ownership

Users retain control and export capability.

Deletion Support

Vault wipe and record deletion mechanisms available.

Portability

Exports use open formats ensuring portability.

Vendor Exposure

No personal data stored remotely by default.

Revision History

Initial compliance summary release.

Memoir.ai — Customer Success Playbook

Version: 1.0

Status: Product & Market Documentation

Last Updated: YYYY-MM-DD

Onboarding Strategy

Guide users through ingestion and timeline exploration.

Retention Strategy

Encourage narrative generation and archive discovery.

Expansion Opportunities

Introduce advanced ingestion and snapshot workflows.

Engagement Monitoring

Track ingestion completion and active usage.

Memoir.ai — Data & Schema Reference Book

Canonical Data Structures, Validation Rules, and Storage Schemas

Document Version: 1.0

Status: Engineering Reference

Owner: Data Architecture Team

Last Updated: YYYY-MM-DD

----

1. Purpose

This reference book defines the canonical data structures, schema rules, validation logic, and persistence models used within Memoir.ai. It serves as the authoritative guide for engineers implementing ingestion, storage, search, AI processing, export, and synchronization systems.

----

2. Scope

This reference includes:

• Canonical event model

• Participant and identity schemas

• Narrative and snapshot schemas

• Provenance and citation models

• Import validation rules

• Deduplication and merge logic

• Normalization pipelines

• Export schemas and packaging

• Supabase metadata schemas

• Row-level security enforcement

• Schema migrations and seeds

User interface behavior is out of scope.

----

3. Canonical Data Model Overview

Memoir.ai standardizes all imported sources into unified structures to enable cross-platform timeline reconstruction and AI summarization.

Core entities include:

• Event

• Person

• Attachment

• Thread

• Narrative

• Citation

• Snapshot

• DataSource

These entities enable ingestion from heterogeneous archives while maintaining consistency.

----

4. Event Entity (Core Record)

The Event is the atomic record within the vault.

Fields include:

event_id — UUID primary key

source_id — Source reference identifier

timestamp — ISO8601 event time

platform — Origin platform identifier

content_raw — Raw message or text body

metadata — Platform-specific structured data

Events may represent messages, posts, media, or emails.

----

5. Person Entity

Participants are extracted and unified across imports.

Fields include:

person_id — UUID identifier

display_name — Preferred name

identities — JSON mapping of emails, phones, handles

Events relate to participants via many-to-many relationships.

----

6. Narrative & Snapshot Entities

Narratives represent AI-generated summaries.

Fields include:

snapshot_id — Snapshot identifier

version_number — Revision count

narrative_body — Markdown narrative text

generated_at — Generation timestamp

Each regeneration creates a new version entry.

----

7. Citation & Provenance Model

Citations map narrative fragments to source evidence.

Attributes include:

citation_id — UUID

target_narrative_id — Narrative reference

source_event_ids — Evidence references

anchor_text — Supported fragment

confidence_score — Evidence strength

Deleting source events invalidates citations.

----

8. Import Validation Rules

Records must pass structural validation before ingestion.

Validation includes:

• Required fields present

• Timestamp format correctness

• Source integrity checks

• Media path verification

• Encoding cleansing

Batches failing >10% validation pause ingestion.

----

9. Deduplication & Merge Logic

Exact duplicates require:

• Timestamp match ±1 second

• Participant match

• Identical content hash

Fuzzy merging applies when:

• Timestamps within ±30 seconds

• Text similarity above threshold

Media deduplicated via SHA-256 hashing.

----

10. Normalization Pipeline

Normalization converts raw platform exports into canonical schema.

Processes include:

• Participant resolution

• Timestamp standardization to UTC

• Event type mapping

• Attachment extraction

• Thread reconstruction

Normalization ensures cross-platform compatibility.

----

11. Export Schema Overview

Exports include events, participants, narratives, and media.

Export structure includes:

export_metadata — Export info

events — Canonical event list

narratives — AI snapshot outputs

citations — Evidence mapping

Exports use open formats to ensure portability.

----

12. Export ZIP Layout

Export bundles follow:

```
Memoir_Export/
  data/
  media/
  docs/
  checksums.txt
```

Media stored via content hash to prevent duplication.

Narratives mirrored as Markdown files for direct readability.

----

13. Supabase Metadata Schema

Cloud layer tracks metadata only.

Primary tables include:

auth_profiles — User identity data

workspaces — Vault ownership records

data_sources — Import sources

import_jobs — Background ingestion jobs

snapshot_records — Snapshot metadata

citations — Evidence metadata

audit_logs — Activity tracking

No personal message content stored remotely.

----

14. Row-Level Security Policies

All Supabase tables enforce RLS.

Rules include:

• Users access only owned workspaces.

• Import jobs restricted by ownership.

• Snapshot access restricted by workspace owner.

Administrative operations restricted to service roles.

----

15. Schema Migrations

Schema evolution handled through ordered migrations.

Migration rules:

• Backward compatibility maintained

• Migration scripts versioned

• Rollback procedures defined

• Checksums verify migration integrity

Production migrations require validation before rollout.

----

16. Development Seeds

Seed data enables development environments.

Seed actions include:

• Insert test users

• Create default workspaces

• Assign free-tier entitlements

Seeds must never include production data.

----

17. Data Integrity Principles

Integrity is preserved through:

• Hash-based deduplication

• Provenance tracking

• Version immutability

• Validation checkpoints

• Recovery support

All ingestion operations must remain idempotent.

----

18. Operational Best Practices

Engineering guidance:

• Validate schema changes early.

• Maintain canonical schema stability.

• Monitor ingestion error ratios.

• Audit deduplication effectiveness.

• Verify export compatibility regularly.

----

19. Conclusion

The Memoir.ai data and schema architecture ensures unified ingestion, accurate provenance tracking, reliable narrative generation, and long-term portability while maintaining user ownership and system integrity.

Memoir.ai — Engineering Runbook

Operational Procedures for Engineering & Platform Teams

Document Version: 1.0

Status: Operational Reference

Owner: Engineering Operations

Last Updated: YYYY-MM-DD

----

1. Purpose

This runbook provides operational procedures for engineering teams responsible for maintaining, deploying, debugging, and recovering Memoir.ai systems. It ensures consistent responses to operational events and platform changes.

----

2. Scope

Covers:

• Development and production environments

• Deployment and rollback operations

• Vault and database recovery

• AI pipeline operations

• Performance troubleshooting

• Logging and monitoring response

• Incident response actions

Does not cover end-user workflows except where operational impact occurs.

----

3. System Architecture Overview

Memoir.ai consists of:

• Electron desktop application shell

• Local encrypted SQLCipher database

• AI processing workers

• Optional Supabase metadata synchronization

• Stripe-based billing and entitlement validation

All sensitive content remains local to user vaults.

----

4. Engineering Responsibilities

Backend Engineers:

• Maintain ingestion and AI pipelines.

• Ensure database integrity and query performance.

• Validate IPC and job execution security.

Frontend Engineers:

• Maintain UI state correctness.

• Handle loading/error states safely.

• Ensure accessibility and performance compliance.

DevOps Engineers:

• Maintain CI/CD reliability.

• Package, sign, and distribute builds.

• Manage release and rollback procedures.

Security Engineers:

• Audit encryption and data handling.

• Review access control mechanisms.

• Monitor dependency vulnerabilities.

----

5. Development Environment Setup

Requirements:

• Node.js v20+

• Package manager configured

• Electron builder installed

• SQLCipher-compatible environment

Setup Flow:

1. Clone repository.

2. Install dependencies.

3. Configure environment variables.

4. Start development environment.

5. Verify vault creation and encryption flow.

----

6. Deployment Procedure

Standard deployment steps:

1. Verify tests pass.
2. Perform dependency security checks.
3. Build UI and backend bundles.
4. Package Electron application.
5. Sign binaries.
6. Publish release artifacts.
7. Trigger update distribution.

Always verify application launch and vault creation post-build.

----

7. Release Gates

Release promotion requires:

• Test suite passes

• No critical vulnerabilities

• Performance targets met

• Manual product review completed

Failure blocks deployment.

----

8. Rollback Procedure

If deployment fails:

1. Pause auto-update channel.

2. Revert release pointer to previous version.

3. Notify users if downgrade required.

4. Validate restored version stability.

Database compatibility must be maintained across versions.

----

9. Database & Vault Recovery

Vault Corruption Response:

1. Attempt integrity check.

2. Trigger index rebuild.

3. Restore from latest backup if needed.

Never modify encrypted DB files manually.

----

10. AI Pipeline Operations

Snapshot Generation Flow:

• Evidence extraction

• Prompt construction

• Local model inference

• Hallucination guard validation

• Citation linking

• Version storage

Failures should retry inference or flag snapshot as incomplete.

----

11. Logging & Diagnostics

Log Levels:

• FATAL: Application crash or DB failure

• WARN: Import or processing issues

• INFO: Job lifecycle events

• DEBUG: Development-only IPC tracing

Logs must not include personal content.

----

12. Monitoring & Alerts

Monitor:

• Disk space

• Worker memory usage

• Database latency

• Job execution times

Alert users for resource exhaustion risks.

----

13. Incident Response

Operational incidents:

1. Identify issue scope.
2. Isolate faulty components.
3. Preserve logs.
4. Apply mitigation.
5. Publish patch if required.

Security incidents:

• Confirm breach possibility.

• Advise vault export and wipe procedures.

• Patch vulnerabilities immediately.

----

14. Performance Troubleshooting

Common issues:

• Slow timeline queries: rebuild indexes.

• Import stalls: verify parsing workers.

• Memory spikes: restart background workers.

• Slow search: confirm indexing completion.

----

15. Secrets & Key Handling

Engineering rules:

• Never store passphrases.

• Never log encryption keys.

• Rotate internal tokens regularly.

• Restrict build secrets to CI environments.

----

16. Backup Recommendations

Recommended procedures:

• Automatic vault backups enabled.

• Users reminded to export periodically.

• Backups stored on separate media.

----

17. Support Interaction Rules

Engineering and support must:

• Never request vault passphrases.

• Never request raw vault content.

• Use sanitized diagnostics only.

----

18. Operational Best Practices

Recommended practices:

• Test migrations locally before release.

• Monitor performance regressions.

• Audit dependencies regularly.

• Validate entitlements gating.

----

19. Conclusion

This runbook standardizes engineering responses and operational execution for Memoir.ai. Adhering to these procedures ensures privacy preservation, reliability, and safe platform evolution.