

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.