Haley.Storage – Full System Specification (Current + New Requirements + Required Modifications)

(Final consolidated write-up for your new session)

---

1. High-Level Purpose

Haley Storage is an enterprise-grade, on-premise document storage subsystem.
It manages:

- Physical storage (Linux LVM strongly recommended)
- Logical metadata via MariaDB
- Multi-level storage hierarchy with:
  Client → Module → Workspace → Directory → File → Version

It supports:

- Controlled IDs, sharding, versioning
- Case sensitivity rules
- Range-based streaming (videos, PDFs, etc.)
- Detailed filesystem/DB synchronization
- Dedicated metadata meta-files (.client.dss.meta, .module.dss.meta, .ws.dss.meta)

This write-up now expands to **Haley.Storage v2**, incorporating new requirements such as:

- External temporary storage (Backblaze B2)
- Chunked uploads
- Placeholder files
- Background sync
- Storage location flags
- Scalable upload flows

---

2. Logical Model (Current)

The storage hierarchy stays the same:

- **Client**
- **Module**
- **Workspace**

- **Directory**
- **File (document + versions)**

Each level has:

- Logical identity (CUID, GUID)
- Physical folder path (except virtual workspaces)
- DB indexing metadata

Sharding rules define file placement to avoid filesystem overload.

Versioning uses:

- document
- doc_version
- version_info
- name_store

---

3. NEW Requirements (Haley.Storage v2)

The system must now support:

3.1 Storage Location Types

- Internal (local disk / LVM)
- External Temporary (e.g., Backblaze B2 during high-load uploads)
- External Permanent (future)

3.2 Supported File States

The system must support placeholder-based lifecycle:

1. Placeholder
2. Upload In Progress
3. Upload Completed
4. Synced to internal storage
5. Error

3.3 External Upload Flow

Must support:

- Creating file record *before* file content is received

- Generating external upload URLs
- Allowing user to upload directly to external storage
- Confirming upload completion
- Immediate usability of external-stored files

3.4 Background Sync

Must provide mechanism to:

- Detect files currently stored externally
- Download them lazily into internal storage
- Update metadata accordingly
- Retry failures

3.5 Chunked Upload Support

Must support:

- Chunked upload sessions
- Chunk indexing
- Resume support
- Chunk completion merging
- Chunked upload for internal & external uploads

3.6 Scalable High-Concurrency Support

Must include:

- Per-user upload concurrency limits
- Global concurrency limits
- Optional upload job queue

3.7 Download Flexibility

Must support:

- Downloading from internal storage
- Downloading from external temporary storage
- Generating short-lived secure URLs
- Handling files not yet synced internally

3.8 Backward Compatibility

- Existing multipart upload endpoints must work unchanged
- Existing download logic must remain functional
- Old files must not require schema migration for reading

---

4. New Metadata & Schema Extensions

Each file (document + version) must support new metadata:

- StorageLocation (Internal, ExternalTemp, ExternalPermanent)
- StatusFlags (bitwise flags)
- ExternalObjectKey
- StorageKey (internal path)
- UploadSessionId
- TemporaryUrl, TemporaryUrlExpiry
- File size
- Hash/checksum
- Sync timestamps

These must be integrated into:

- document
- doc_version
- version_info

---

5. Current System Summary (Before Modifications)

(This section is from your existing write-up, summarized so new session has context)

5.1 Upload Flow

- Upload is strictly multipart/form-data
- Upload writes directly to disk immediately
- DB indexing commits after file is physically written
- Controlled by ProcessAndBuildStoragePath
- Versioning handled via doc_version/version_info
- No external or placeholder support

5.2 Download + Streaming

- All downloads assume file is internal
- Reads FileStream directly from internal path
- Range support built manually or via ASP.NET Core pipeline

5.3 Directory, Workspace, Module, Client Registration

- Creates physical folders
- Indexes DB entries
- Writes metadata .dss.meta files
- Sharding rules create deep folder paths

5.4 Indexer Responsibilities

- Maintaining all DB entities
- Path resolution
- Version info updates
- Transaction handling across multiple DBs

---

6. Required Modifications (High-Level)

(This section is critical: WHAT must change to meet the new requirements)

6.1 Upload Pipeline (Upload(IOSSWrite)) Must Change

Must be extended to:

- Allow placeholder-only records
- Bypass physical disk write for external upload mode
- Support chunked upload session creation
- Support session-based byte append
- Accept metadata-only „complete upload” signals
- Tag file with ExternalTemp status if needed
- Finalize version info without disk write when external

6.2 Path Resolution (ProcessAndBuildStoragePath) Must Change

Currently assumes:

- Every file corresponds to a local path
- Sharding must always be run
- Folder must exist before upload

It must support:

- External keys (not filesystem paths)
- Placeholder entries without paths
- Deferred path generation
- Decoupled sharding (only needed after final sync)

6.3 Indexer Schema Support Must Change

Indexer must now handle:

- Storing StorageLocation
- Storing StatusFlags
- Recording external object keys
- Storing upload session IDs
- Supporting file entries with no internal path yet
- Update status after sync

6.4 Download Logic Must Change

Download must:

- Detect internal vs external
- Generate signed URLs for external-temp files
- Support immediate redirect/download
- Handle non-local read streams
- Provide “not yet synced” fallback logic

6.5 Delete Flow Must Change

Delete must:

- Delete external objects (if present)
- Delete internal file
- Safely handle placeholders
- Support deletion before sync

6.6 Streaming Controller Must Change

Must support:

- Non-local streams
- Skipping range logic if external redirect is used
- Conditional range support for internal files only

6.7 Background Sync Worker Must Integrate

New worker must:

- Query files marked ExternalTemp + UploadCompleted
- Download to internal storage path
- Commit internal version_info update
- Update status flags

Storage service must expose:

- Query APIs
- Update status APIs
- Safe version merge handlers

6.8 Storage Backend Abstraction Must Be Introduced

Currently everything is tied to filesystem I/O.
New design must allow:

- InternalStorageBackend (local disk)
- ExternalStorageBackend (Backblaze)
- Future custom backends

The service must remain backend-agnostic.

---

7. Compatibility Rules

1. The **existing multipart endpoint remains untouched**.
2. Existing uploads still write to disk as before.
3. Existing downloads continue working.
4. New external/chunked flows do NOT break any part of old behavior.
5. Old files do not require migration.

---

8. Final Summary

Haley.Storage v2 must be redesigned to introduce:

- External temporary storage support
- Placeholder + state-flag lifecycle
- Chunked uploads
- Background syncing
- Storage abstraction layer
- Status-aware path resolution
- Scalable upload concurrency
- Secure external download logic

All while maintaining:

- The existing file hierarchy
- Existing metadata structure
- Existing upload/download flows
- Backward compatibility

This document contains all you need to upload to a new session to continue the implementation plan.