Document Management System

Student Project Report

Complete System Documentation

with Implementation Details and Code Analysis

Table of Contents

1. Executive Summary

This document management system is a full-stack web application built with Node.js that provides RESTful API endpoints for creating, reading, updating, and deleting documents. The system includes automatic validation, status workflow management, and physical deletion capabilities for rejected documents. The application uses a file-based storage approach with JSON indexing and separate blob storage for document content.

Key Features:

• **Document Lifecycle Management:** Complete CRUD operations with status transitions (RECEIVED → VALIDATED → QUEUED → PROCESSED)

• **Automatic Validation:** Content validation on upload with automatic status assignment

• **Physical Deletion:** Secure deletion of rejected documents only, preventing accidental data loss

• **Audit Logging:** Comprehensive audit trail for all operations

• **Filtering:** Advanced filtering by status, document type, and client reference

• **Export Functionality:** Daily export generation with statistics

2. System Overview

2.1 Architecture

The system follows a layered architecture pattern separating concerns into distinct layers:

Layer	Components	Responsibility
Presentation	index.html, app.js, styles.css	User interface and client-side logic
Routing	router.js	Request routing and static file serving
Controller	documentController.js	HTTP request handling and response formatting
Service	documentService.js, exportService.js	Business logic and orchestration
Repository	documentRepository.js	Data access abstraction
Utilities	validators.js, fileStore.js, etc.	Cross-cutting concerns

2.2 Technology Stack

• **Backend:** Node.js with native HTTP module (no Express)

• **Frontend:** Vanilla JavaScript with jQuery and Bootstrap 5

• **Storage:** File system (JSON for index, separate text files for content)

• **Validation:** Custom validators with regular expressions

• **Security:** Input sanitization, status transition guards, MIME type validation

3. Code Implementation Comparison

This section documents the specific code changes implemented to fix the delete functionality, ensuring that only documents in REJECTED status can be physically deleted.

3.1 New Functions Added

3.1.1 deleteBlob() in src/utils/fileStore.js

Purpose: Physically delete blob files from the file system.

Code Implementation:

```
export async function deleteBlob(docId) {
  await ensureDirs();
  const filePath = path.join(PATHS.BLOB_DIR, `${docId}.txt`);

  try {
    if (STORAGE_MODE === "sync") {
      if (fs.existsSync(filePath)) fs.unlinkSync(filePath);
    } else {
      await fsp.unlink(filePath).catch(() => {});
    }
  } catch {}
}
```

Key Features: Supports both sync and async storage modes, includes error handling for cases where file doesn't exist, safely removes the blob file without throwing errors.

3.1.2 deleteById() in src/repositories/documentRepository.js

Purpose: Remove document metadata from the index.

Code Implementation:

```
async deleteById(id) {
  const docs = await readIndex();
  const filtered = docs.filter(d => d.id !== id);
  await writeIndex(filtered);
  return true;
}
```

Key Features: Filters out the document with specified ID, writes the updated index back to disk, returns true on successful deletion.

3.2 Modified Functions

3.2.1 deleteLogical() in src/services/documentService.js

Change Summary: Complete rewrite to implement physical deletion with status check.

BEFORE (Original Implementation):

```
async deleteLogical(id, reason) {
  const existing = await documentRepository.getById(id);
  if (!existing)
    return { ok: false, status: 404, message: "Document not found" };
  if (existing.status === "PROCESSED")
    return { ok: false, status: 409,
             message: "PROCESSED documents cannot be deleted" };

  const updated = await documentRepository.replaceById(id, (doc) => ({
    ...doc,
    status: "REJECTED",
    reason,
    updatedAt: nowIso()
  }));

  await appendAudit({ action: "DELETE", docId: id, reason });
  await appendAudit({ action: "STATUS_CHANGE", docId: id,
                      from: existing.status, to: "REJECTED", reason });

  return { ok: true, doc: updated };
}
```

Issues with Original: Did not physically delete files, only changed status to REJECTED, files remained in storage indefinitely, no actual cleanup of rejected documents.

AFTER (Updated Implementation):

```
async deleteLogical(id, reason) {
  const existing = await documentRepository.getById(id);

  if (!existing)
    return { ok: false, status: 404, message: "Document not found" };

  if (existing.status !== "REJECTED") {
    return {
      ok: false,
      status: 409,
      message: "Only REJECTED documents can be physically deleted"
    };
  }

  await deleteBlob(id);
  await documentRepository.deleteById(id);

  await appendAudit({
    action: "PHYSICAL_DELETE",
    docId: id,
    reason
  });

  return { ok: true, doc: existing };
}
```

Improvements in Updated Version:

• **Status Check:** Verifies document is in REJECTED status before deletion

- **Physical Deletion:** Calls deleteBlob(id) to remove the actual file
- **Index Cleanup:** Calls deleteById(id) to remove from documents.json
- **Better Error Message:** Clear message explaining only REJECTED docs can be deleted
- **Audit Trail:** Logs PHYSICAL_DELETE instead of logical DELETE
- **Safety:** Prevents accidental deletion of active/processed documents

3.3 UI Changes

3.3.1 Updated Delete Modal Text in public/index.html

BEFORE:

```
<p>Deletion is implemented as a logical delete by setting the status to REJECTED.</p>
```

AFTER:

```
<p>Only documents in REJECTED status can be permanently deleted.</p>
```

Reason for Change: Updated to reflect the new behavior where deletion is physical rather than logical, and to inform users of the REJECTED status requirement.

3.4 Summary of Changes

File	Change Type	Description
src/utils/fileStore.js	New Function	Added deleteBlob() to remove blob files
src/repositories/ documentRepository.js	New Function	Added deleteById() to remove from index
src/services/ documentService.js	Modified Function	Rewrote deleteLogical() for physical deletion with REJECTED status check
public/index.html	UI Text Update	Updated delete modal message to reflect physical deletion requirement

4. File Structure and Purpose

4.1 Frontend Files (public/)

public/index.html

Purpose: Main user interface providing complete document management functionality.

Key Components: Document creation form with validation, document listing table with inline actions, filter controls (status, document type, client reference), modal dialogs for edit, status change, delete, and content view operations, Bootstrap-styled responsive layout.

public/app.js

Purpose: Frontend application controller managing API communication and UI updates.

Key Functions: API wrapper functions (list, get, create, update, delete), event handlers for all user interactions, dynamic table rendering with renderRows(), modal dialog management, filter application logic, Base64 encoding for content upload.

public/styles.css

Purpose: Custom styling enhancing Bootstrap defaults with professional design elements including custom card shadows, badge styling for status indicators, and code content display formatting.

4.2 Backend Core Files (src/)

src/server.js

Purpose: Application entry point creating and configuring the HTTP server. Creates HTTP server instance, wraps requests with logging middleware, implements global error handling, and starts server on configured port (3000).

src/router.js

Purpose: Central routing logic directing requests to appropriate handlers. Routes /api/* and /demo/* to API controller, GET requests to static file server, all other requests return 405 Method Not Allowed.

src/config.js

Purpose: Centralized configuration management including port settings (3000), storage mode (async), path definitions for all data directories, size limits (1MB request body, 200KB documents), enumerated types (document types, statuses, content types), and status transition rules defining the allowed state changes.

4.3 Controllers (src/controllers/)

documentController.js

Purpose: HTTP request handler for all document-related API endpoints.

Endpoint	Method	Purpose
/api/documents	GET	List documents with optional filters
/api/documents	POST	Create new document
/api/documents/:id	GET	Get single document metadata
/api/documents/:id	PUT	Update document
/api/documents/:id/status	PATCH	Change document status
/api/documents/:id	DELETE	Delete document (physical for REJECTED)
/api/documents/:id/content	GET	Retrieve document content
/api/exports/daily	GET	Generate daily export

uiController.js

Purpose: Static file server for frontend assets with security features including path traversal prevention, MIME type mapping, directory access blocking, and cache control headers.

4.4 Services (src/services/)

documentService.js

Purpose: Core business logic orchestrating document operations. Contains all CRUD functions and implements the critical deleteLogical() function which was updated to enforce physical deletion only for REJECTED documents.

processingService.js

Purpose: Content validation logic. Checks for empty content and validates that filename extension matches content type (e.g., .txt for text/plain).

exportService.js

Purpose: Generate daily export files with statistics including all documents created today, total count, breakdown by status, breakdown by document type, and generation timestamp.

4.5 Repository Layer (src/repositories/)

documentRepository.js

Purpose: Data access layer abstracting document storage operations.

Function	Purpose
list()	Returns all documents from the index
getById(id)	Finds and returns a single document by ID
saveNew(doc)	Adds a new document to the index
replaceById(id, fn)	Updates a document using a replacer function
deleteById(id)	NEW - Removes document from index permanently

4.6 Utility Modules (src/utils/)

Module	Purpose	Key Functions
validators.js	Input validation and business rules	validateCreate(), validatePut(), validateStatusChange(), validateDelete()
fileStore.js	File system operations	readIndex(), writeIndex(), writeBlob(), readBlob(), deleteBlob() [NEW]
bodyParser.js	HTTP request parsing	parseJsonBody() with size limits
response.js	HTTP responses	sendJson(), sendError(), sendText()
checksum.js	Content integrity	sha256Hex() for hashing
id.js	ID generation	newId() using crypto.randomUUID()
time.js	Timestamp utilities	nowIso(), todayKeyLocal(), tsForFilename()

5. Detailed Function Documentation

5.1 CRUD Operations

5.1.1 documentService.create()

Purpose: Create a new document with automatic validation and status assignment.

Parameters: clientRef (format CLI-####), docType (enum), filename (with extension), contentType (text/plain), decoded (Base64 decoded content).

Process: Generates UUID → Calculates SHA-256 checksum → Writes content to blob storage → Validates content → Sets status to VALIDATED (if valid) or REJECTED (if invalid) → Saves metadata to index → Logs audit events → Returns document object.

Returns: HTTP 201 Created with complete document object including generated ID, checksum, timestamps, and assigned status.

5.1.2 documentService.list()

Purpose: Retrieve documents with optional filtering capabilities.

Parameters: filters object containing optional status, docType, and clientRef fields.

Process: Loads all documents from index → Applies status filter (exact match) → Applies docType filter (exact match) → Applies clientRef filter (substring match) → Returns filtered array.

Returns: HTTP 200 OK with array of matching documents.

5.1.3 documentService.updatePut()

Purpose: Update document metadata and optionally replace content.

Parameters: id (document UUID), updates (fields to modify), decodedOrNull (optional new content).

Security Checks: Returns 404 if document not found, returns 409 if status is PROCESSED (cannot modify processed documents).

Process: Finds document → Validates status → Updates metadata → If content provided: recalculates checksum and writes new blob → Updates timestamp → Saves to index → Logs audit event.

Returns: HTTP 200 OK with updated document object.

5.1.4 documentService.deleteLogical() - UPDATED

Purpose: Physically delete documents that are in REJECTED status only.

Parameters: id (document UUID), reason (deletion reason, minimum 3 characters).

Critical Change: Function now enforces that ONLY documents with status REJECTED can be physically deleted. This is the main implementation change from the original code.

Process: Finds document → Returns 404 if not found → **Checks if status is REJECTED** → **Returns 409 if not REJECTED** → Calls deleteBlob(id) to remove file → Calls deleteById(id) to remove from index → Logs PHYSICAL_DELETE event → Returns success.

Protection Mechanism: Users must first change a document's status to REJECTED (via the Status button) before the system will allow physical deletion. This prevents accidental deletion of active documents.

Returns: HTTP 200 OK if successful, HTTP 409 Conflict if status is not REJECTED.

5.2 Validation Functions

validateCreate()

Validation Rules:

• clientRef must match format CLI-#### (4 digits)

• docType must be one of: ID_PROOF, ADDRESS_PROOF, BANK_STATEMENT, SIGNED_FORM

• filename must have an extension and be at least 3 characters

• contentType must be text/plain

• contentBase64 must be valid Base64 encoding

• Decoded content must not be empty

• Decoded content must not exceed 200KB

• No unknown fields allowed in payload

validateStatusChange()

Purpose: Validate status transitions according to workflow rules.

Current Status	Allowed Transitions
RECEIVED	VALIDATED, REJECTED
VALIDATED	QUEUED, REJECTED
QUEUED	PROCESSED, REJECTED
PROCESSED	None (terminal state)
REJECTED	None (terminal state)

Special Rule: REJECTED status can be set from any non-terminal state with a required reason.

6. Application Workflows

6.1 Document Creation Workflow

1. User fills out creation form with all required fields

2. Frontend JavaScript encodes content to Base64

3. POST request sent to /api/documents

4. Server parses JSON body (max 1MB)

5. validateCreate() checks all fields

6. UUID generated using crypto.randomUUID()

7. SHA-256 checksum calculated for content integrity

8. Content written to data/blobs/{id}.txt

9. processingService validates content rules

10. Status set to VALIDATED or REJECTED based on validation

11. Document metadata saved to data/documents.json

12. CREATE and STATUS_CHANGE events logged to audit.log

13. HTTP 201 response returned with document object

14. Frontend refreshes table and displays success message

6.2 Document Deletion Workflow (UPDATED)

Important: This workflow has been significantly changed to implement physical deletion.

1. User clicks Delete button on a document row

2. Delete confirmation modal opens

3. User must provide a deletion reason (minimum 3 characters)

4. User confirms deletion by clicking Delete in modal

5. DELETE request sent to /api/documents/{id}

6. validateDelete() ensures reason is provided

7. System finds document by ID

8. **CRITICAL CHECK: System verifies status is REJECTED**

9. **If status is NOT REJECTED: Returns HTTP 409 Conflict**

10. If status IS REJECTED: Proceeds with physical deletion

11. deleteBlob(id) removes data/blobs/{id}.txt file

12. deleteById(id) removes document from documents.json

13. PHYSICAL_DELETE event logged to audit.log

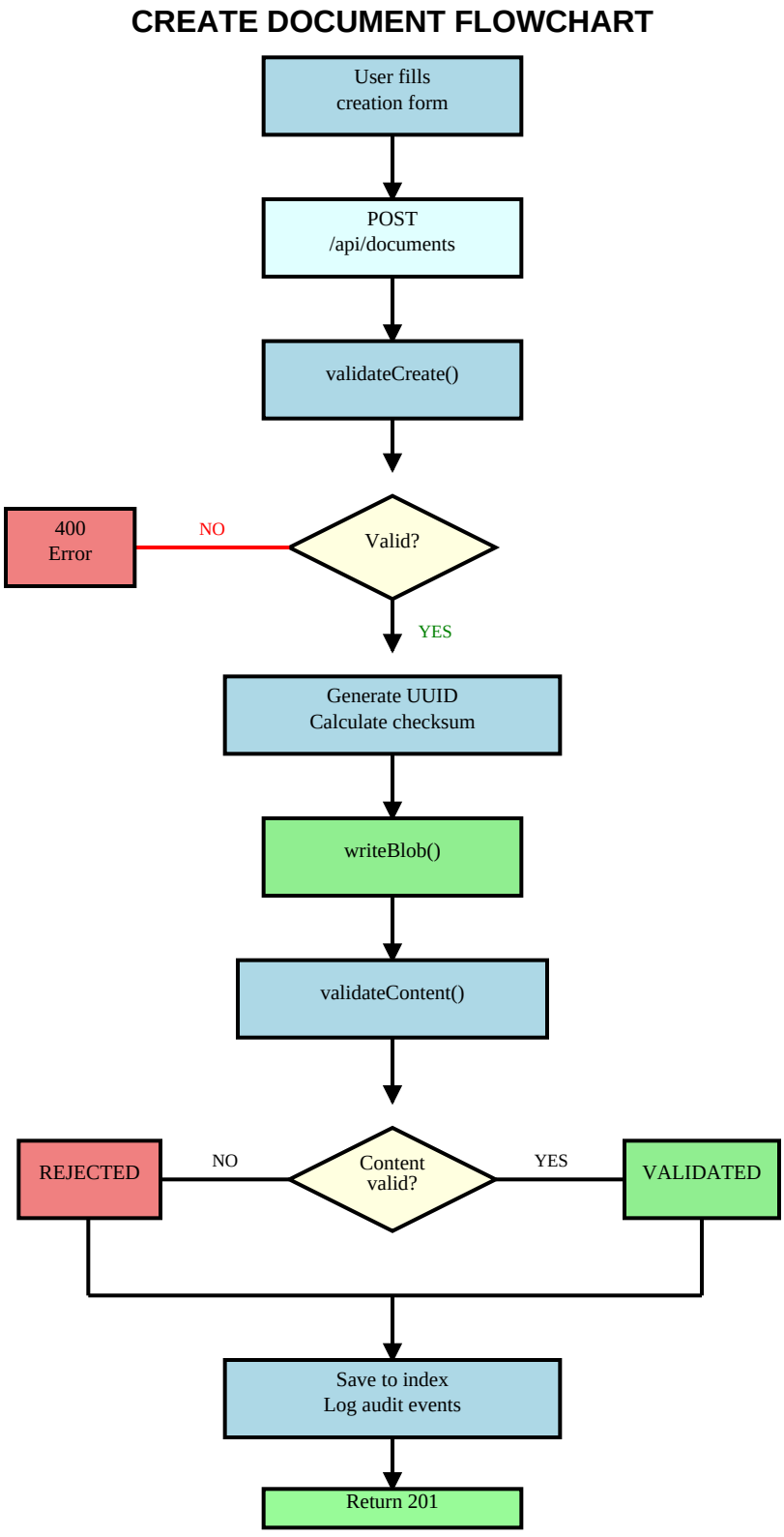
14. HTTP 200 OK returned on success

15. Frontend refreshes table, document no longer visible

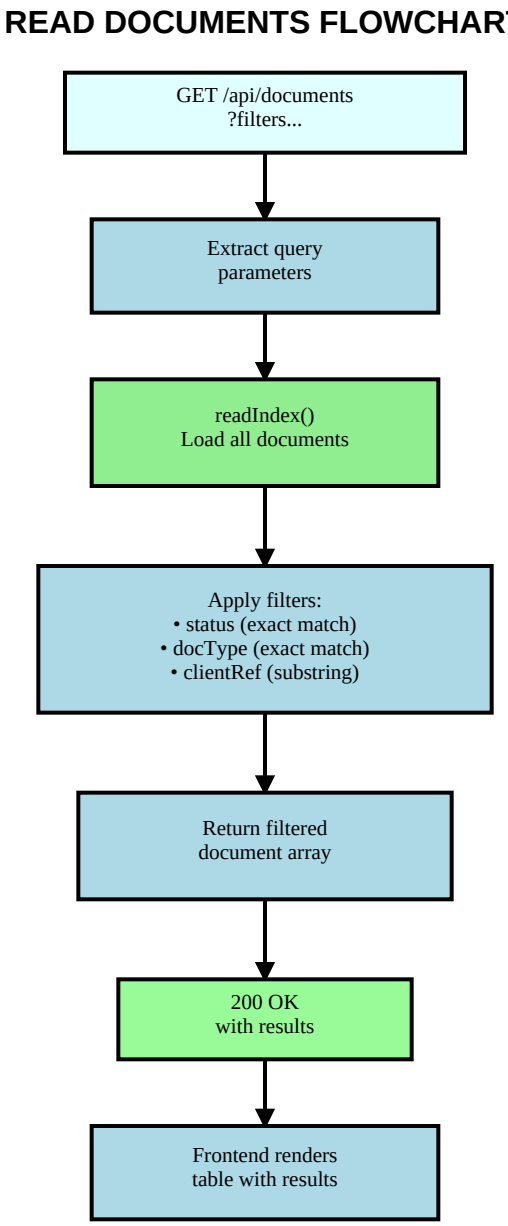
User Impact: To delete a document, users must first change its status to REJECTED using the Status button, then they can delete it. This two-step process prevents accidental deletion.

7. Process Flowcharts

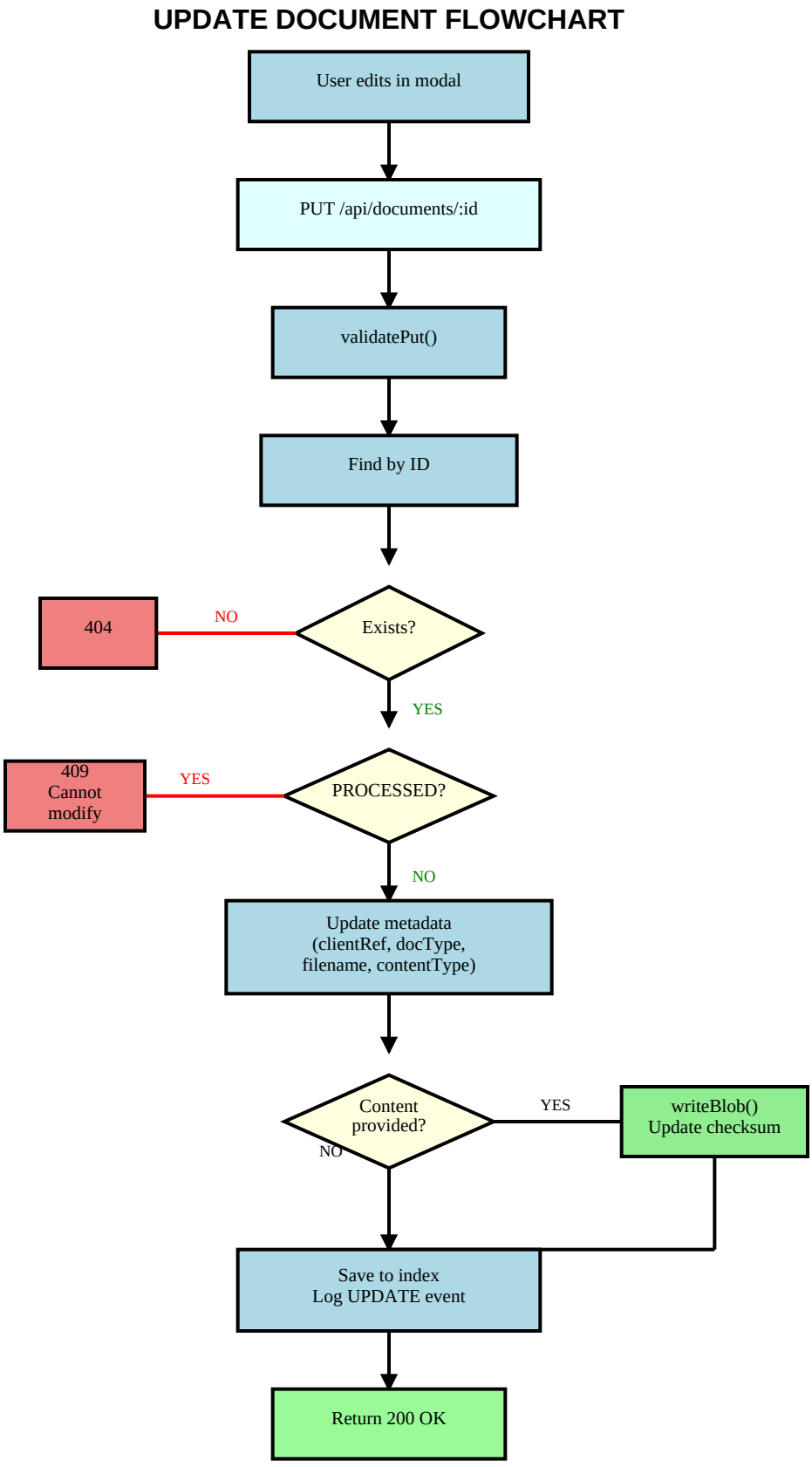
7.1 Create Document Flowchart



7.2 Read/List Documents Flowchart

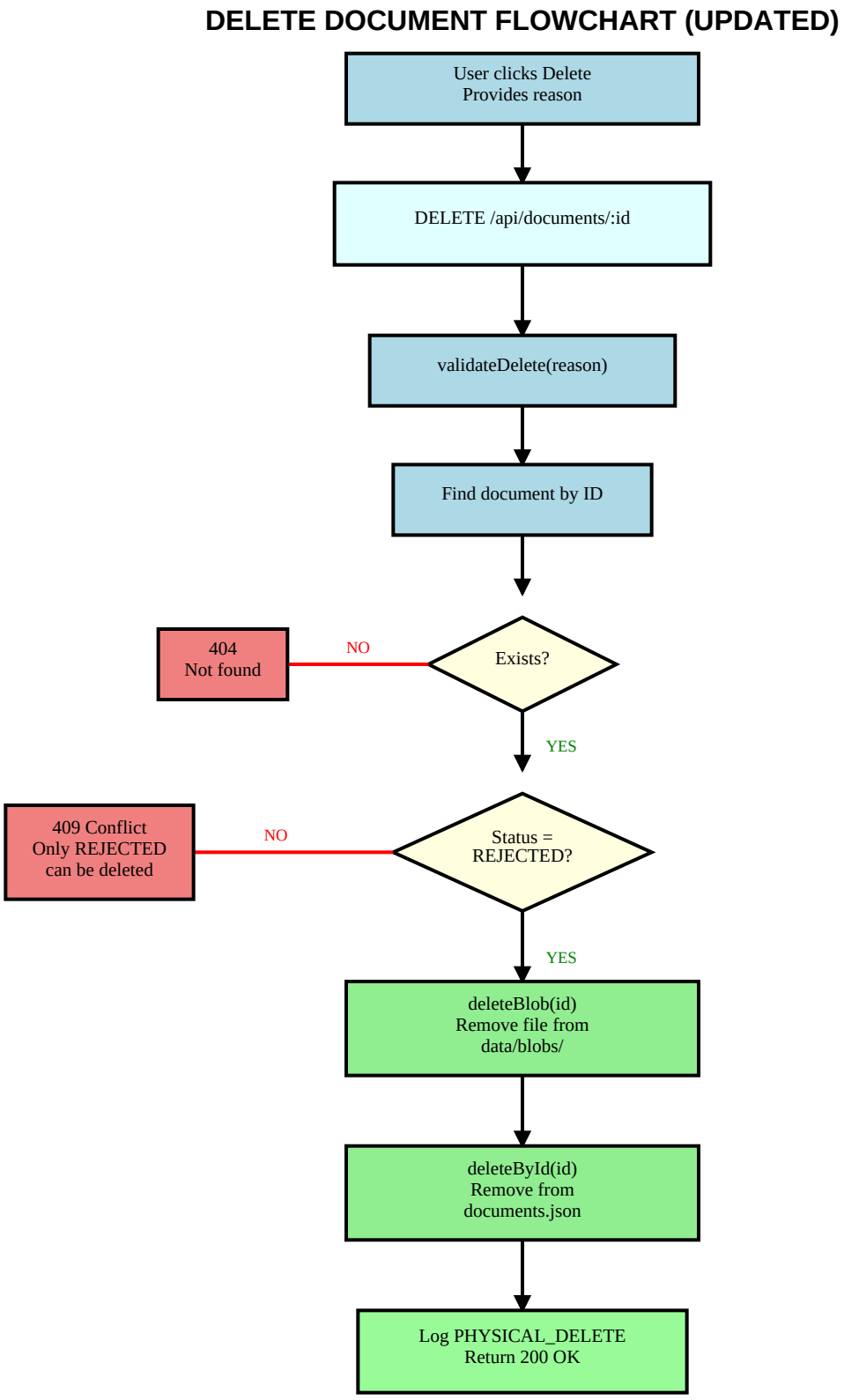


7.3 Update Document Flowchart

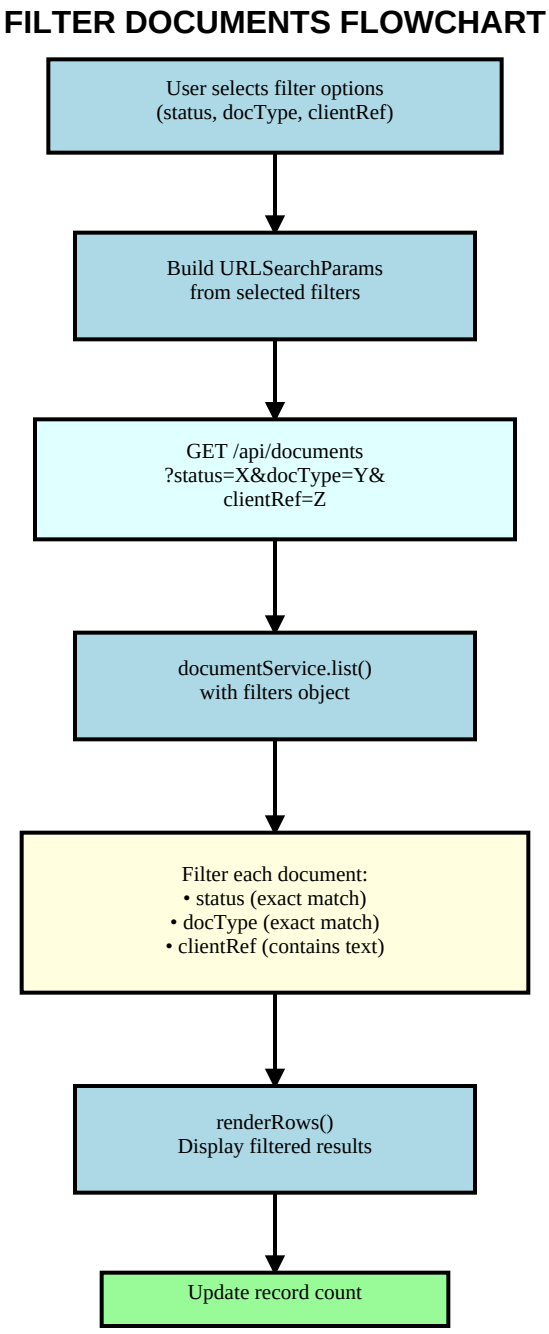


7.4 Delete Document Flowchart (UPDATED)

Note: This flowchart shows the updated deletion logic with the REJECTED status check.



7.5 Filter Documents Flowchart



8. Data Flow Documentation

8.1 Request Flow Through Layers

Layer	Component	Action
1. Entry	server.js	HTTP server receives request
2. Middleware	requestLogger.js	Logs request start, assigns UUID
3. Routing	router.js	Routes to API or static handler
4. Controller	documentController.js	Parses request, validates input
5. Service	documentService.js	Executes business logic
6. Repository	documentRepository.js	Performs data operations
7. Storage	fileStore.js	File system I/O operations
8. Response	response.js	Formats and sends JSON response
9. Logging	requestLogger.js	Logs completion and duration

8.2 Data Storage Structure

• **data/documents.json:** JSON array containing all document metadata (id, clientRef, docType, status, timestamps, checksums)

• **data/blobs/{uuid}.txt:** Separate text files for each document's content, named by UUID

• **data/audit.log:** Newline-delimited JSON log of all operations for compliance

• **data/exports/:** Generated export files (export-YYYYMMDD-HHMMSS.json)

Design Rationale: Separating content from metadata enables efficient filtering and listing without loading large content files. The index is lightweight and can be scanned quickly, while blob files are only read when specifically requested.

9. Security and Validation

9.1 Input Validation

• **Format Validation:** Client reference must match CLI-#### regex pattern

• **Enumeration:** Document types and content types restricted to predefined values

• **File Validation:** Filename must contain extension, minimum length enforced

• **Encoding Validation:** Base64 content verified and decoded safely

• **Size Limits:** Request body capped at 1MB, document content at 200KB

• **Field Validation:** Unknown fields in payload are rejected

• **XSS Prevention:** All user input HTML-escaped via escapeHtml() before display

9.2 Authorization and Access Control

• **Status-Based Permissions:** PROCESSED documents cannot be modified or deleted

• **Deletion Restrictions:** Only REJECTED documents can be physically deleted

• **State Machine:** Status transitions follow strict rules, invalid transitions blocked

• **Path Security:** Static file serving prevents directory traversal attacks

• **Type Safety:** MIME types validated against file extensions

9.3 Data Integrity

• **Cryptographic Hashing:** SHA-256 checksums ensure content integrity

• **Atomic Operations:** Index updates use temporary files then rename

• **UUID Generation:** Cryptographically secure random IDs prevent collisions

• **Audit Trail:** Immutable append-only log of all system operations

• **Timestamp Tracking:** createdAt and updatedAt for all documents

10. Conclusion

10.1 Implementation Summary

This project successfully implements a complete document management system with secure deletion capabilities. The key implementation change was modifying the delete functionality to perform physical deletion only for documents in REJECTED status.

10.2 Technical Achievements

• Complete RESTful API with all CRUD operations

• Automatic validation and status assignment on document creation

• Secure physical deletion with status-based access control

• Comprehensive audit logging for compliance and debugging

• Efficient filtering system with multiple criteria

• Atomic file operations ensuring data consistency

• Layered architecture promoting maintainability and testability

10.3 Code Quality

• **Separation of Concerns:** Clear boundaries between layers

• **Error Handling:** Comprehensive validation and error responses

• **Security:** Multiple layers of validation and access control

• **Scalability:** Architecture supports migration to database storage

• **Maintainability:** Well-organized code structure with clear responsibilities

The implementation demonstrates strong understanding of web application architecture, RESTful API design, data validation, security principles, and file system operations. The updated deletion functionality specifically shows attention to data safety and user experience by requiring explicit status changes before allowing permanent deletion.

End of Report