Scanner App — Requirements & Architecture Specification

Version: 1.0
Date: 2026-02-07

1. Purpose & Vision

This document captures the complete requirements and architectural decisions for a simple but extensible iOS Scanner application. The intent is to deliver a production■quality v1 scanner app while deliberately structuring the codebase so the scanning functionality can be reused inside future, more complex applications without refactoring.

2. Core User Goals

Users should be able to quickly scan physical documents using their device camera, correct perspective, combine multiple pages into a single document, save the result as a PDF, and access those documents across devices via iCloud.

3. Functional Requirements

• Multi■page scanning sessions
• Automatic document detection and perspective correction
• Manual crop true■up per page
• Page rotation
• Review screen supporting reorder, delete, and retake page
• Save finalized document as PDF in app sandbox
• Share and print via iOS share sheet
• Save to Files on demand
• Library view with thumbnails and document state indicators
• PDF viewer with pinch■to■zoom and page navigation
• Settings screen with 3■tier output presets
• CloudKit sync for scans and settings (private database)

4. Draft & Document Lifecycle

Scanning begins by creating a locally persisted draft document. Drafts are visible in the library and may be resumed later. A draft becomes a finalized document only when the user explicitly saves, at which point a PDF is generated and eligible for CloudKit sync.

Document States:
• draft – local only, resumable, no PDF
• savedLocal – PDF generated locally
• syncing – CloudKit upload in progress
• synced – fully synchronized
• syncError – local saved, CloudKit failed (retryable)

5. Storage Architecture

All scanner content is stored inside Application Support to avoid user■visible clutter. Each document owns its own directory, enabling atomic operations and clean draft handling.

Application Support/Scanner/
• Documents/<UUID>/
  • metadata.json
  • thumbnail.jpg
  • pages/001.jpg …
  • output/document.pdf
• Settings/settings.json

6. CloudKit Design

CloudKit uses the private database only. Each finalized document maps one■to■one with a CloudKit record, using the document UUID as the record name.

Record Types:
• ScanDocument – metadata + PDF CKAsset + thumbnail asset
• ScannerSettings – singleton record synced across devices

7. Architecture & Packaging

The system is split into two repositories to eliminate future refactoring.

ScannerKit (Swift Package, separate GitHub repo):
• Camera capture coordination
• Crop & perspective correction
• Draft persistence
• PDF generation
• Core models and storage logic
• Reusable SwiftUI components

ScannerApp (iOS App):
• Navigation and UI shell
• Library and settings UI
• CloudKit implementation
• ScannerKit dependency via Git URL

8. Technology Decisions

• SwiftUI■first UI architecture
• UIKit/VisionKit only where required for scanning
• PDFKit for viewing
• CloudKit private database only
• Three output presets (Small / Balanced / High)

9. Identifiers

• Package: ScannerKit
• App Bundle ID: com.DavidMWilcox.ScannerApp
• CloudKit Container: iCloud.com.DavidMWilcox.ScannerApp
• Git Host: GitHub

10. MVP Scope (v1)

The v1 release includes scanning, review, PDF export, library browsing, settings, and CloudKit sync. OCR, tagging, sharing, and advanced organization are explicitly out of scope.