MerchTools

Software Design Document

Charles Clark

7 November 2025

Table of Contents

1. INTRODUCTION 3

1.1 PURPOSE 3
1.2 SCOPE 3
1.3 OVERVIEW 3
1.4 REFERENCE MATERIAL 5
1.5 DEFINITIONS AND ACRONYMS 5

2. SYSTEM OVERVIEW 5

3. SYSTEM ARCHITECTURE 5

3.1 ARCHITECTURAL DESIGN 6
3.1.1 Use Case ID: UC-01 10
3.1.2 Use Case ID: UC-02 10
3.1.3 Use Case ID: UC-03 11
3.1.4 Use Case ID: UC-04 12
3.1.5 Use Case ID: UC-05 13
3.1.6 Use Case ID: UC-06 13
3.1.7 Use Case ID: UC-07 14
3.2 DECOMPOSITION DESCRIPTION 15
3.2.1 UC-01 Scan SKU 15
3.2.2 UC-02 Search SKU 16
3.2.3 UC-03 Add SKU 17
3.2.4 UC-04 Add Audit Item 18
3.2.5 UC-05 Generate Report 19
3.2.6 UC-06 Share Report 20
3.2.7 UC-07 View History 21
3.3 DESIGN RATIONALE 21

4. DATA DESIGN 22

4.1 DATA DESCRIPTION 22
4.2 DATA DICTIONARY 22

5. COMPONENT DESIGN 23

6. HUMAN INTERFACE DESIGN 25

6.1 OVERVIEW OF USER INTERFACE 25
6.2 SCREEN IMAGES 26
6.3 SCREEN OBJECTS AND ACTIONS 28

7. REQUIREMENTS MATRIX 28

8. APPENDICES 29

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system of the mobile (Android) application "MerchTools." The intended audience for this SDD includes:

- **Instructors & Reviewers**
- **Developers & Software Architects**
- **QA Testers**
- **Maintainers & Support Team Members**
- **Other Interested Parties & Stakeholders**

1.2 Scope

This SDD describes MerchTools as a standalone, offline-first Android application. MerchTools unify scanning/searching, note taking, photo capture, and PDF reporting in a single, simple workflow. Business goals: faster escalation and correction of inventory/shelf issues, clear accountability, and portable reports that optimize audit processes.

1.3 Overview

This SDD details the system, architecture, and design of MerchTools and is organized into sections and subsections in the following manner:

1. **Introduction**
2. **System Overview –** General description of the functionality, context, and design of MerchTools
3. **System Architecture**
    a. **Architectural Design –** High-level overview of how responsibilities of the system were partitioned and then assigned to subsystems for system functionality.
    b. **Decomposition Description –** Illustrating the logical steps through the program.
    c. **Design Rationale**
4. **Data Design**

   a. **Data Description –** Description of the storing/processing/organization of major data/system entities.
   b. **Data Dictionary –** Alphabetical list of major data/system entities with associated types and descriptions.

5. **Component Design –** Systemic description of individual components using a procedural description language (PDL) or pseudocode.
6. **Human Interface Design**
   a. **Overview of User Interface –** Description of the functionality of the system from the user's perspective.
   b. **Screen Images –** UI screenshots.
   c. **Screen Objects and Actions –** Description of screen objects and associated actions.
7. **Requirements Matrix –** Cross-reference of components to the software requirements specification (SRS) document.
8. **Appendices**

1.4 Reference Material

1.5 Definitions and Acronyms

i. **Audit:** A short review of items on a shelf/section with counts and photos.
ii. **UPC:** Universal Product Code for identifying products.
iii. **EAN:** European Article Number for identifying products (compatible with UPCs).
iv. **GTIN:** Global Trade Item Number found under the barcode on a product's packaging.
v. **SKU:** Stock Keeping Unit (specific product).
vi. **MVVM:** Model-View-ViewModel; software architecture that separates an application into three interconnected components to enhance maintainability, testability, and organization.
    - **View:** The view is responsible for defining the structure, layout, and appearance of what the user sees on screen.
    - **ViewModel:** The view model implements properties and commands to which the view can data bind to, and notifies the view of any state changes through change notification events.
    - **Model:** Model classes are non-visual classes that encapsulate the app's data.
vii. **Compose/Jetpack Compose:** Android's recommended modern toolkit for building native UI; simplifies and accelerates UI development with less code, powerful tools, and intuitive Kotlin APIs.

2. SYSTEM OVERVIEW

MerchTools is an Android app (Kotlin, Jetpack Compose, MVVM) that optimizes shelf audits & inventory management with a simple workflow: Scan/Search → Add count/note/photo → Repeat for items → Generate PDF → Share → History.

All data is stored locally using Room; reports (PDF/CSV) are generated using Android PdfDocument (or HTML-PDF via WebView print framework as a fallback) and can be shared using Android Sharesheet (email, text messaging, etc.).

This product emphasizes reliability in poor connectivity, simple UX, data integrity, performance, and privacy (no cloud dependency; offline except sharing reports).

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

Styles & Patterns: MVVM + Clean Architecture + Repository. Offline-first with Room for data persistence. Use Cases/Interactors encapsulate app actions; PDF/CSV generation is a local service using Android PdfDocument; sharing uses Android Sharesheet/Intent; media (photos) are stored as files referenced by DB paths/URIs.

High-level subsystems & responsibilities

a. **Scanner & Search**
   i. Camera barcode/UPC scanning (via CameraX and ML Kit)
   ii. Manual text search across local SKU index

b. **Add SKU**
   i. Manual entry (with help of barcode scanning to populate some fields) for adding SKU to local SKU index

c. **Audit Capture**
   i. Create/manage Audit and AuditItems
   ii. Validation guardrails (counts, required fields)
   iii. Photo capture & attachment (optional)

d. **Report Generator**
   i. PDF composition with detailed Audit list page(s), timestamp, metadata (Android PdfDocument)
   ii. Optional CSV report (feature pending)

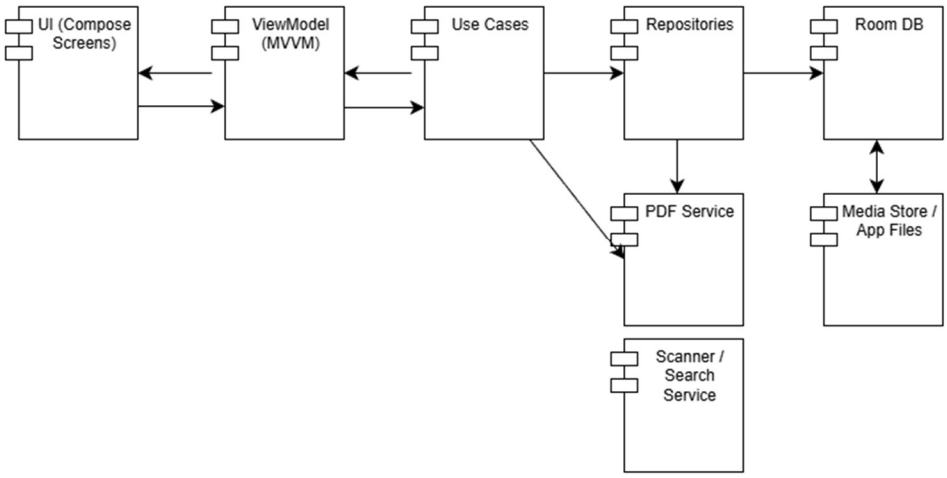
e. **Share**
   i. Send report via email, text message, etc., using Android Sharesheet invocation

f. **History (feature pending)**
   i. Audit Report archiving and History listing

g. **Data Layer**
   i. Room entities/DAOs for Store, Section, Audit, AuditItem, Photo, Sku
   ii. Repository implementations for coordinating actions

h. **Platform Services**
   i. App permissions, Camera, File I/O, Time

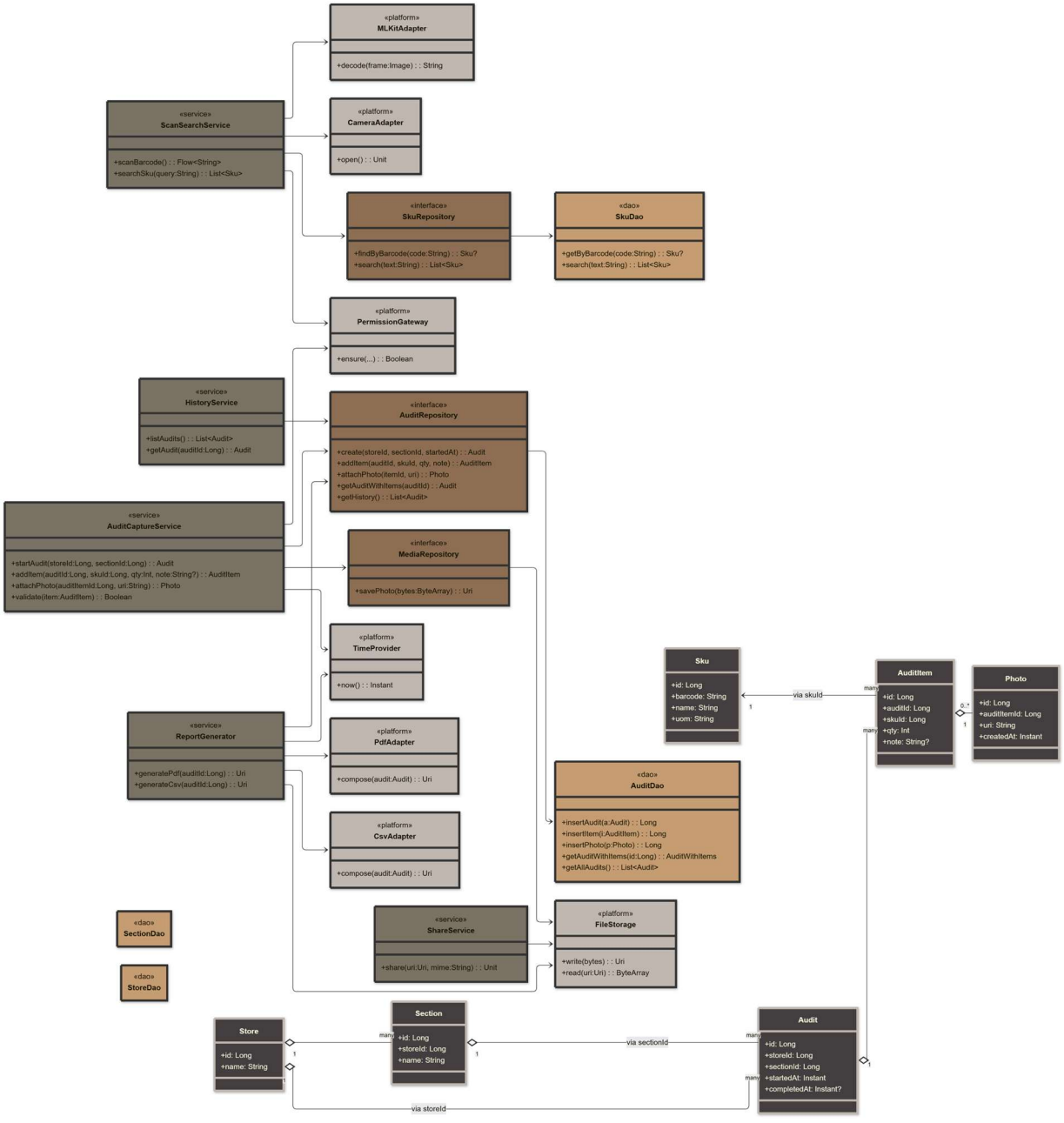
Module Relationships



UML Class Diagram: Interconnections between major subsystems and data repositories

service

The following UML class diagram illustrates—at a high-level—how the major subsystems (scanner/search, audit capture, report generator, share, history, data layer, platform services) and data repositories collaborate for system functionality.


«platform»
MLKitAdapter
+decode(frame:Image) : : String
«service»
ScanSearchService
+scanBarcode() : : Flow<String>
+searchSku(query:String) : : List<Sku>
«platform»
CameraAdapter
+open() : : Unit
«interface»
SkuRepository
+findByBarcode(code:String) : : Sku?
+search(text:String) : : List<Sku>
«dao»
SkuDao
+getByBarcode(code:String) : : Sku?
+search(text:String) : : List<Sku>
«platform»
PermissionGateway
+ensure(...) : : Boolean
«service»
HistoryService
+listAudits() : : List<Audit>
+getAudit(auditId:Long) : : Audit
«interface»
AuditRepository
+create(storeId, sectionId, startedAt) : : Audit
+addItem(auditId, skuId, qty, note) : : AuditItem
+attachPhoto(itemId, uri) : : Photo
+getAuditWithItems(auditId) : : Audit
+getHistory() : : List<Audit>
«service»
AuditCaptureService
+startAudit(storeId:Long, sectionId:Long) : : Audit
+addItem(auditId:Long, skuId:Long, qty:Int, note:String?) : : AuditItem
+attachPhoto(auditItemId:Long, uri:String) : : Photo
+validate(item:AuditItem) : : Boolean
«interface»
MediaRepository
+savePhoto(bytes:ByteArray) : : Uri
«platform»
TimeProvider
+now() : : Instant
Sku
+id: Long
+barcode: String
+name: String
+uom: String
via skuId
1
many
AuditItem
+id: Long
+auditId: Long
+skuId: Long
+qty: Int
+note: String?
0..*
1
Photo
+id: Long
+auditItemId: Long
+uri: String
+createdAt: Instant
«service»
ReportGenerator
+generatePdf(auditId:Long) : : Uri
+generateCsv(auditId:Long) : : Uri
«platform»
PdfAdapter
+compose(audit:Audit) : : Uri
«dao»
AuditDao
+insertAudit(a:Audit) : : Long
+insertItem(i:AuditItem) : : Long
+insertPhoto(p:Photo) : : Long
+getAuditWithItems(id:Long) : : AuditWithItems
+getAllAudits() : : List<Audit>
«platform»
CsvAdapter
+compose(audit:Audit) : : Uri
«dao»
SectionDao
«dao»
StoreDao
«service»
ShareService
+share(uri:Uri, mime:String) : : Unit
«platform»
FileStorage
+write(bytes) : : Uri
+read(uri:Uri) : : ByteArray
Store
+id: Long
+name: String
Section
+id: Long
+storeId: Long
+name: String
via sectionId
Audit
+id: Long
+storeId: Long
+sectionId: Long
+startedAt: Instant
+completedAt: Instant?
via storeId

This class diagram abstracts UI (Compose screens/ViewModels) to focus on subsystems + data. Subsystems call repositories; repositories call DAOs; DAOs persist entities. Platform adapters/services hide Android specifics.

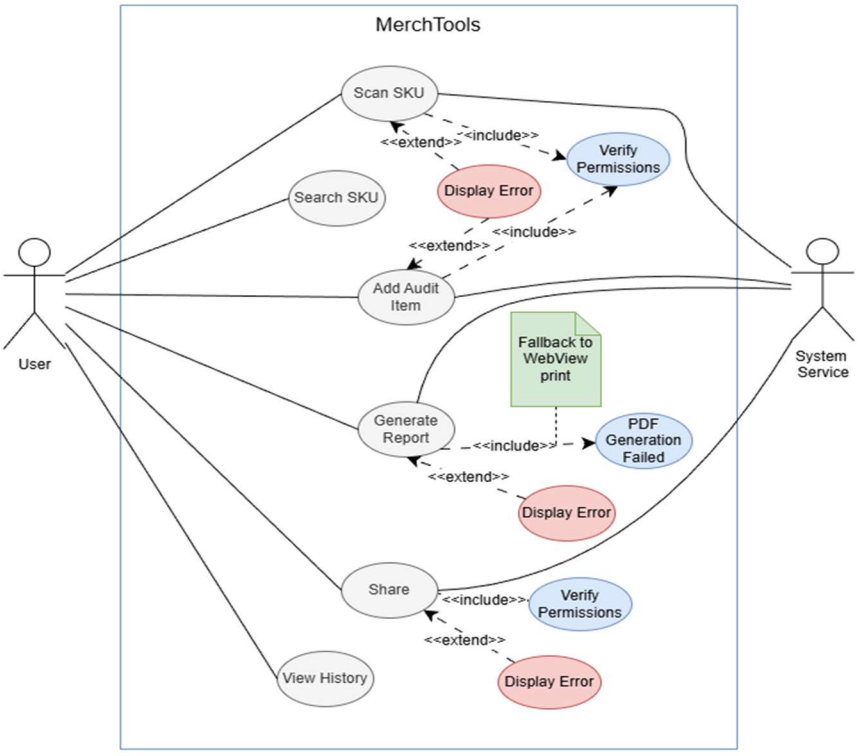
Use Case Modeling

Actors: User (primary), System Services (e.g., Camera, Storage, File APIs, etc.) (secondary)

Essential Use Cases

1. **Scan SKU** (scan UPC/barcode to identify SKU)
2. **Search SKU** (manual text search)
3. **Add SKU** (add SKU to catalog)
4. **Add Audit Item** (enter count, note, attach photo)
5. **Generate Report** (PDF with optional CSV (feature pending))
6. **Share Report** (email/share via intent)
7. **View History** (past audits)

System Use Case Diagram



3.1.1 Use Case ID: *UC-01*

Use Case Name: Scan SKU

Relevant Requirements: FR-3.1 (scan/search item)

Primary Actor: User

Pre-conditions: Application running; application has camera permission; SKU catalog locally available.

Post-conditions: SKU is selected; matching item is displayed; or user is notified no match found and offers manual search.

Basic Flow or Main Scenario:

1. User taps Scan
2. System opens camera and scans barcode/UPC
3. System decodes value using ML Kit Barcodescanner
4. If one match, system selects that SKU and navigates to **Add Item**
5. If no match, system offers manual search

Extensions or Alternate Flows:

a. Camera permissions denied – show rationale and route to settings or use manual search

Exceptions:

a. Scan fails (poor lighting) – show retry UI

b. Decode error – display error and show retry UI, fallback to manual

Related Use Cases: UC-02 (Search SKU), UC-03 (Add Audit Item)

3.1.2 Use Case ID: UC-02

Use Case Name: Search SKU

Relevant Requirements: FR-3.1 (scan/search item)

Primary Actor: User

Pre-conditions: Application running; SKU catalog locally available.

Post-conditions: SKU is selected; matching item is displayed; or no SKU found message is displayed; offer add SKU.

Basic Flow or Main Scenario:

1. User taps Search
2. System opens SKU catalog with search box

3. User types in SKU/UPC/brand/item name
4. System displays relevant search results
5. User taps on correct SKU
6. System navigates to **Add Item**
7. If no match, system displays no item found and offers add SKU

Extensions or Alternate Flows:

a. Previous Scan attempt failed; manual search offered
b. User taps manual search
c. System opens catalog/search; user enters search

Exceptions:

a. Search fails (no item found) – show add SKU UI

Related Use Cases: UC-03 (Add Audit Item)

3.1.3 Use Case ID: UC-03

Use Case Name: Add SKU

Relevant Requirements: FR-3.6

Primary Actor: User

Pre-conditions: Application running; no SKU found from Search or Scan; no SKU in catalog.

Post-conditions: SKU successfully entered into Catalog.

Basic Flow or Main Scenario:

1. User taps Add SKU
2. System navigates to Add SKU screen; displays SKU fields (SKU, UPC, brand, size) for single addition
3. User taps Scan
4. System opens camera and scans barcode/UPC
5. System decodes value using ML Kit Barcodescanner
6. System populates fields
7. User enters brand, size; taps Add
8. System validates required fields; successfully enters SKU into Catalog

Extensions or Alternate Flows:

a. Previous Scan/Search attempts result in no match
b. System prompts user to Add SKU

c. User taps Add SKU

d. System navigates to Add SKU screen

Exceptions:

a. Scan unable to decode values – display rationale and navigate back to Add SKU screen for complete manual entry

b. Addition unsuccessful – display error message

c. Duplicate SKU found upon Add attempt – display rationale

Related Use Cases: UC-01 (Scan SKU), UC-02 (Search SKU)

3.1.4 Use Case ID: UC-04

Use Case Name: Add Audit Item

Relevant Requirements: FR-3.2 (record counts/notes/photos)

Primary Actor: User

Pre-conditions: Scan results in a valid match; or Search results in a valid match.

Post-conditions: SKU/item with count (required), note (optional), and photo (optional) added successfully to current (or new) report.

Basic Flow or Main Scenario:

1. System displays Add Audit Item screen with relevant SKU details (number/UPC/name/type)
2. User enters integer count
3. User enters note (optional)
4. User attaches photo (optional)
5. System displays resized, attached photo under Photo field (if applicable)
6. User taps Add Item
7. System validates fields
8. If validation succeeds; SKU/item is added to current/new report
9. System displays add more UI

Extensions or Alternate Flows:

a. User leaves note and/or photo empty

b. System adds item with only count recorded

c. User taps add photo from gallery

d. System opens local gallery, if storage permissions are granted

Exceptions:

a. User enters invalid count (validation fail)
b. System displays relevant error
c. Storage permissions denied – show rationale and route to settings or go back to Add Audit Item screen

Related Use Cases: UC-01, UC-02, UC-03 (Add SKU), UC-05 (Generate Report)

3.1.5 Use Case ID: UC-05

Use Case Name: Generate Report

Relevant Requirements: FR-3.3

Primary Actor: User

Pre-conditions: Open/current audit with at least 1 SKU/item

Post-conditions: Generated timestamped PDF summarizing an audit with listed SKUs and associated recorded counts/notes/photos; success message displayed; share button displayed.

Basic Flow or Main Scenario:

1. User taps Generate Report
2. System generates PDF report from audit and displays success (or fail) message
3. System displays generated report for view with Share button in the bottom screen

Extensions or Alternate Flows:

a. User closes out of displayed generated report
b. System navigates to home screen
c. User closes opened audit (before generating report)
d. System saves draft and navigates to home screen

Exceptions:

a. Generate Report fails – display error message and try again UI

Related Use Cases: UC-04 (Add Audit Item), UC-06 (Share Report)

3.1.6 Use Case ID: UC-06

Use Case Name: Share Report

Relevant Requirements: FR-3.4

Pre-conditions: Successful generation of PDF report from audit.

Post-conditions: System opens share methods via Android Sharesheet/intent; system navigates to home screen.

Basic Flow or Main Scenario:

1. User taps Share
2. System opens share methods dialog via Sharesheet/intent (e.g., email, text message, etc.)
3. User taps desired method
4. System opens selected method with PDF report attached
5. System navigates to home screen

Extensions or Alternate Flows:

a. User selects report to share from View History

Exceptions:

a. Permissions denied – show rationale and route to settings or navigate back to previous screen

Related Use Cases: UC-05 (Generate Report), UC-07 (View History)

3.1.7 Use Case ID: UC-07

Use Case Name: View History

Relevant Requirements: FR-3.5

Primary Actor: User

Pre-conditions: Application running; past audits available in History list.

Post-conditions: System navigates to selected audit in History list and displays the completed report.

Basic Flow or Main Scenario:

1. User taps View History
2. System displays list of past/completed audits and associated generated reports
3. User selects audit from list
4. System displays selected audit with listed SKUs and associated recorded counts/notes/photos
5. User selects Share or closes out of audit
6. System opens share methods via Sharesheet/intent; or navigates back to View History screen

Extensions or Alternate Flows:

a. No past audits; system displays no audits found

Exceptions:

a. Unable to open selected audit – display error message and go back to View History screen

Related Use Cases: UC-01 (Scan SKU), UC-02 (Search SKU), UC-05 (Generate Report), UC-06 (Share Report)

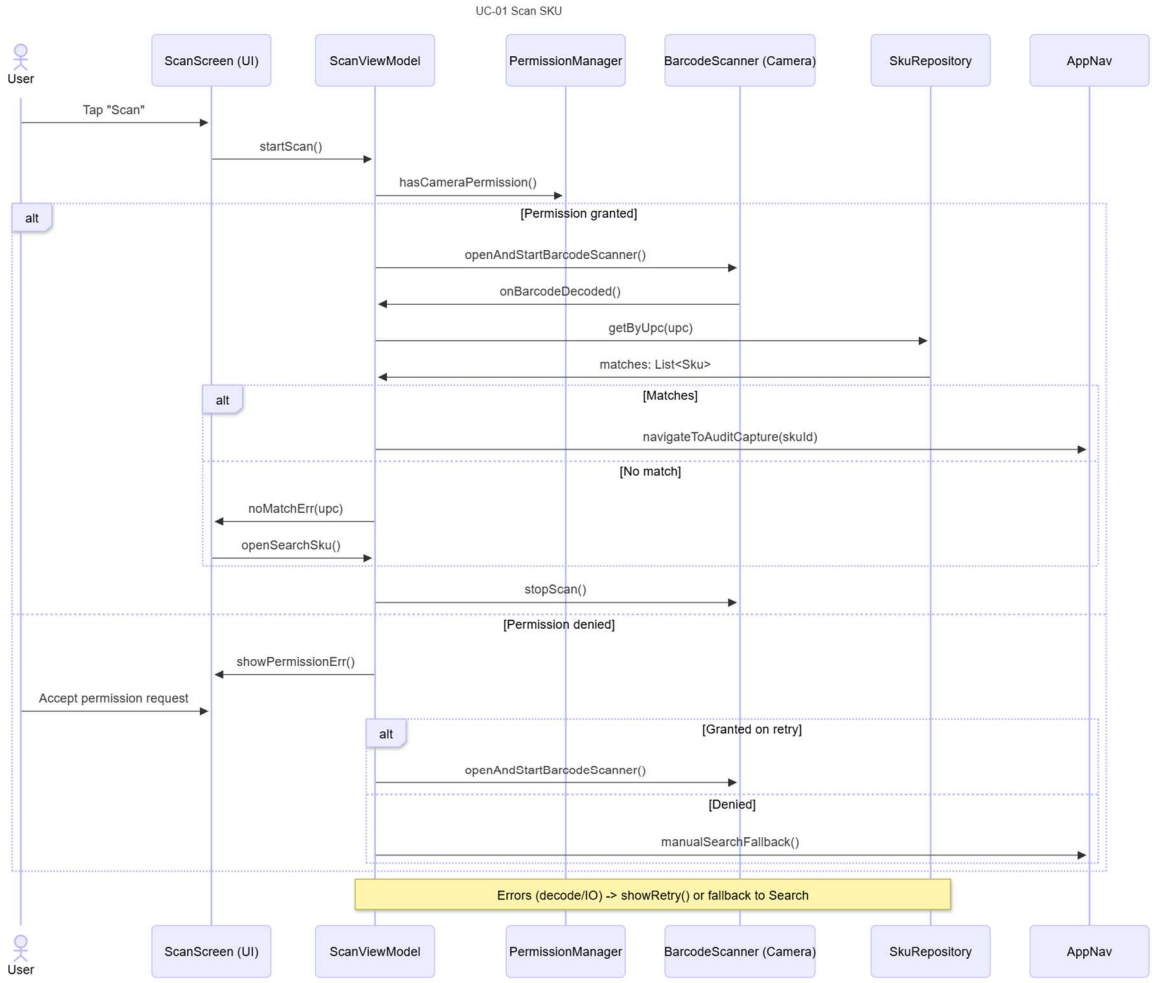
3.2 Decomposition Description

3.2.1 UC-01 Scan SKU

Decomposition (objectives and responsibilities)

- **User / ScanScreen:** initiates scan; displays permission prompts and results
- **ScanViewModel:** delegates permission check, scanner start/stop, SKU lookup, and navigation
- **PermissionManager:** checks/requests Camera permission
- **BarcodeScanner:** opens camera, decodes UPC, returns string(s)
- **SkuRepository:** resolves decoded UPC to SKU from local DB
- **AppNav:** routes to **AuditCapture** when SKU is matched

Sequence Diagram



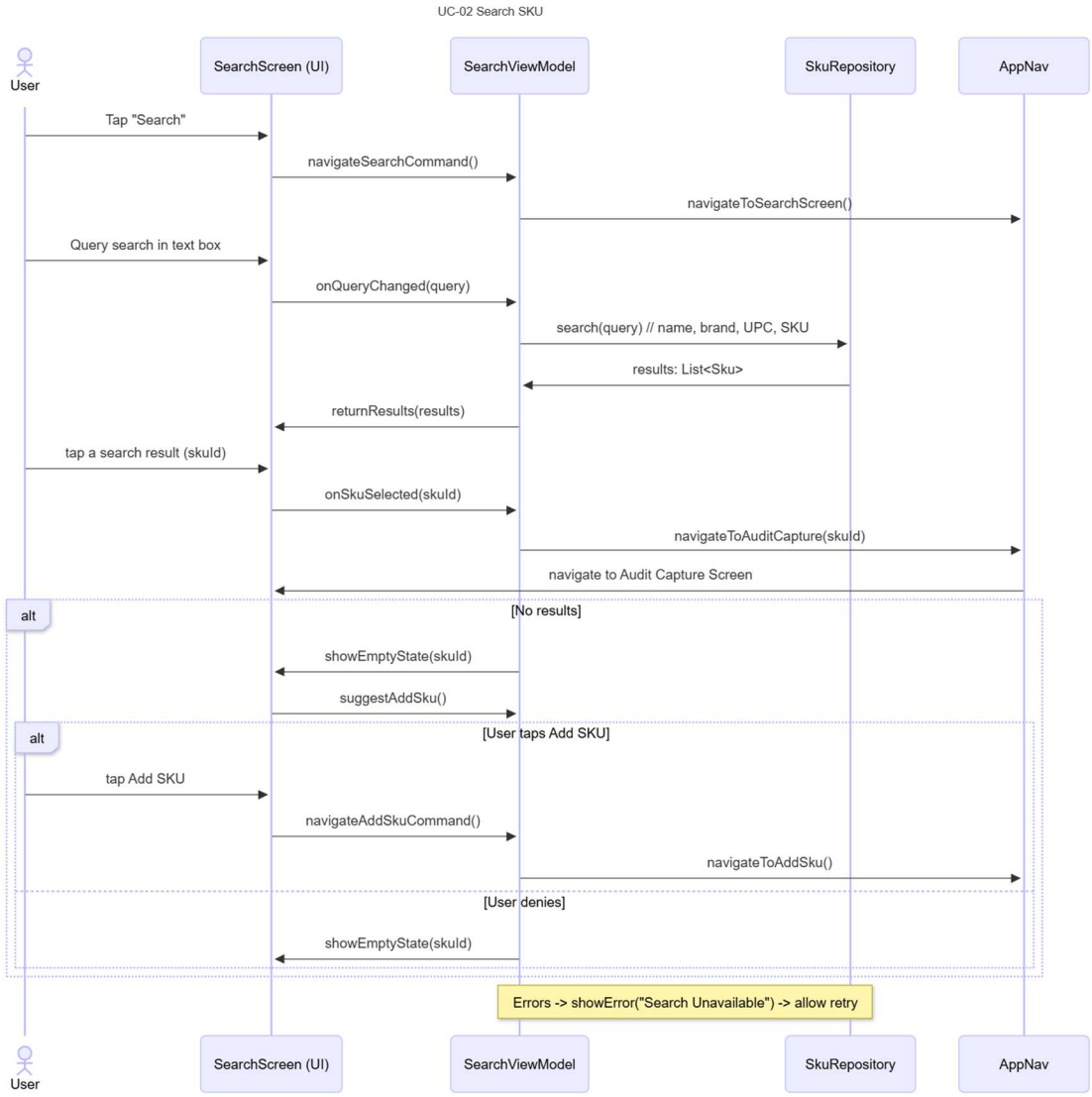
3.2.2 UC-02 Search SKU

Decomposition (objectives and responsibilities)

- **User / SearchScreen:** initiates search; displays SKU catalog and search box
- **SearchViewModel:** delegates commands, SKU index search, and navigation
- **SkuRepository:** returns search results from local DB that matches or is close to User input

  AppNav: routes to **AuditCapture** when SKU is selected

Sequence Diagram

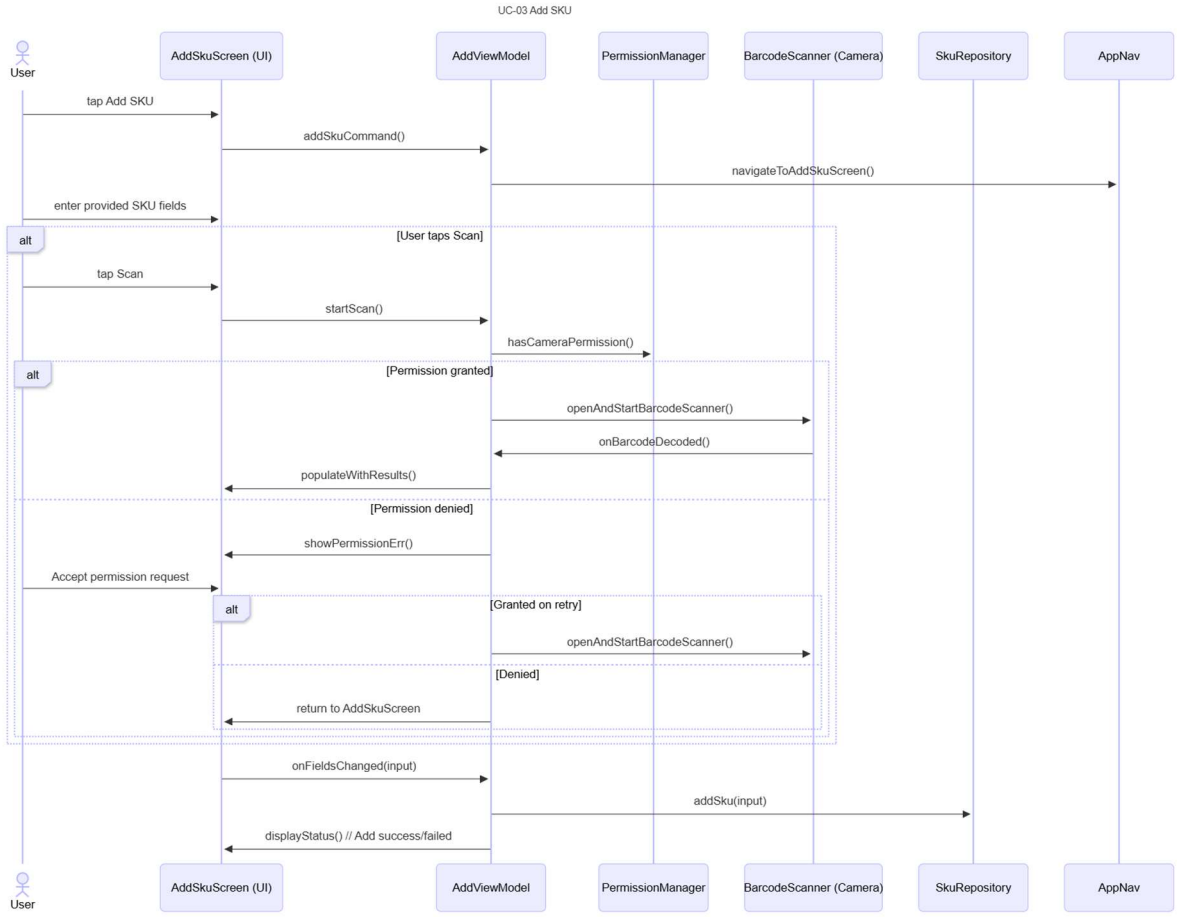


3.2.3 UC-03 Add SKU

Decomposition (objectives and responsibilities)

- **User / AddSkuScreen:** initiates add SKU; displays SKU fields for addition
- **PermissionManager:** checks/requests Camera permission
- **BarcodeScanner:** opens camera, decodes UPC, returns string(s)
- **SkuRepository:** enters SKU into SKU catalog

Sequence Diagram

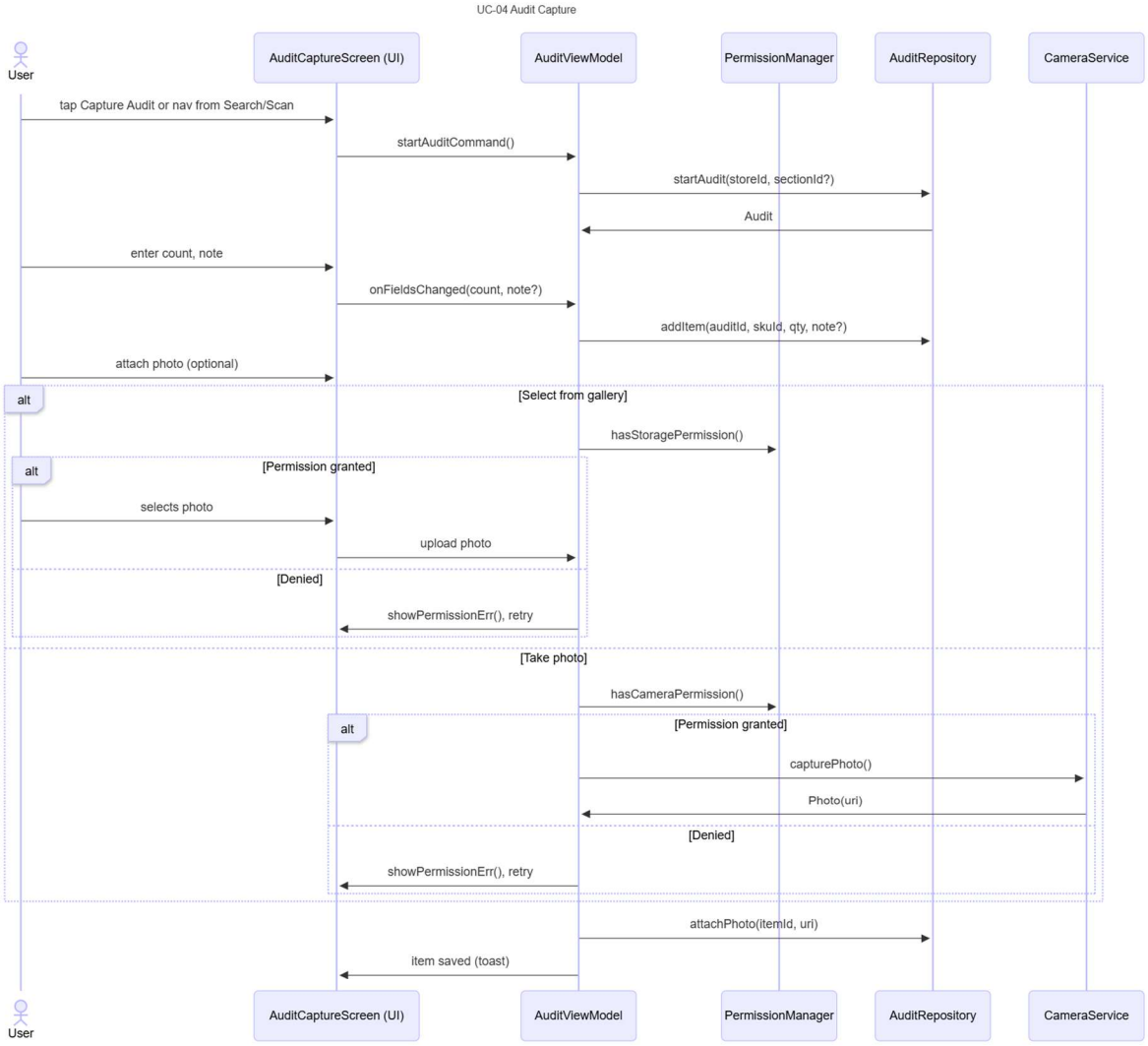


3.2.4 UC-04 Add Audit Item

Decomposition (objectives and responsibilities)

- **User / AuditCaptureScreen:** initiates audit; adds first…*n* SKU to new or open audit
- **AuditViewModel:** checks for open audit; validates fields
- **AuditRepository:** interface for creating audit, adding items, attaching photos, getting past or current audit, and getting audit history
- **PermissionManager:** checks/requests Camera permission

Sequence Diagram

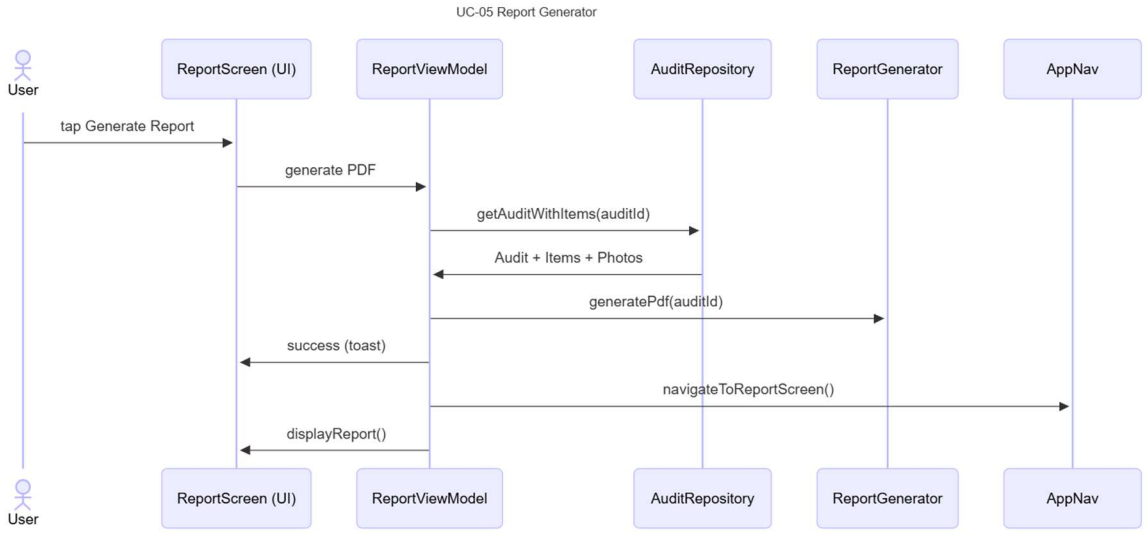


3.2.5 UC-05 Generate Report

Decomposition (objectives and responsibilities)

- **User / ReportViewModel:** initiates generate report
- **ReportGenerator:** generates timestamped PDF report with SKU details
- **ReportScreen:** displays generated report
- **AuditRepository:** fetches completed audit with items
- **AppNav:** routes to ReportScreen after generation

Sequence Diagram

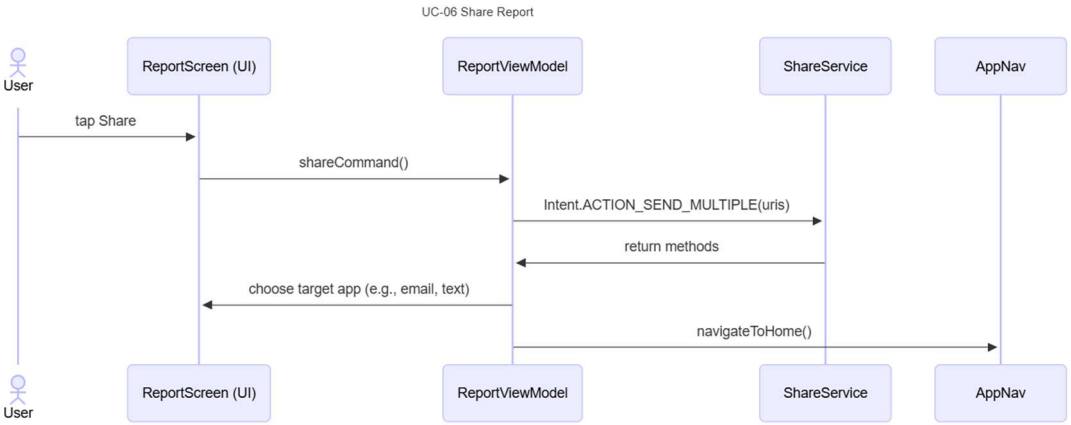


3.2.6 UC-06 Share Report

Decomposition (objectives and responsibilities)

- **User / ReportScreen:** view report; initiate share via email, text, etc.
- **ReportViewModel:** share command
- **ShareService:** Android Sharesheet/intents to target desired method

Sequence Diagram

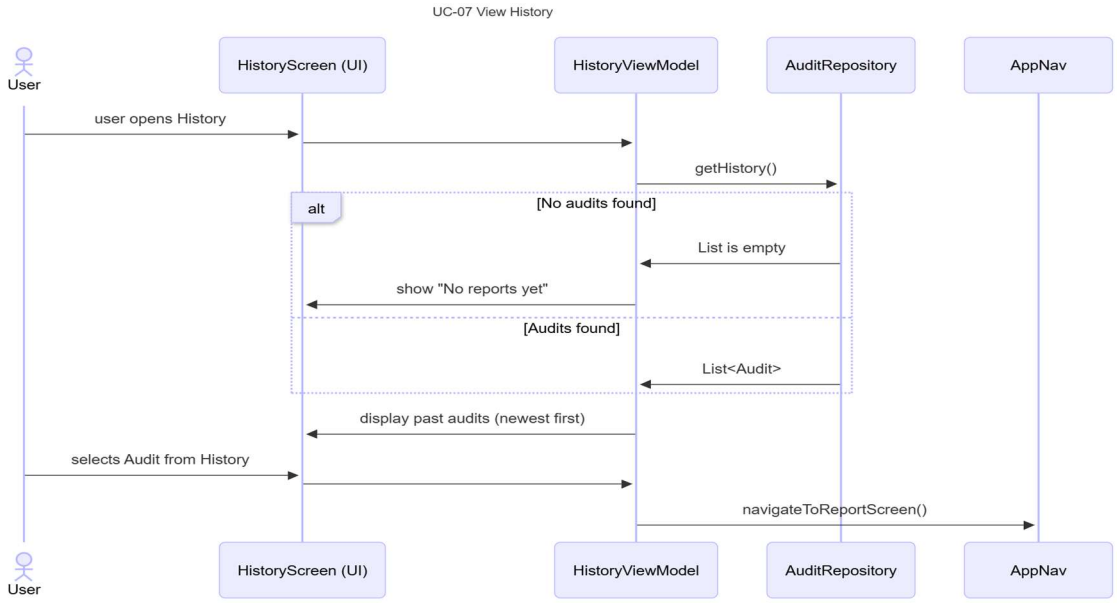


3.2.7 UC-07 View History

Decomposition (objectives and responsibilities)

- **User / HistoryScreen:** opens list of completed audits (newest first)
- **HistoryViewModel:** loads AuditRepository.getHistory()
- **AuditRepository:** returns `List<Audit>`
- **AppNav:** History → ReportScreen (view) or Share

Sequence Diagram



3.3 Design Rationale

Architecture Choice: Following MVVM + Clean architecture improves testability, maintainability, and separation of concerns: Compose screens bind to ViewModels; use cases encapsulate business logic. Additionally, all core flows are completed on-device without relying on the network supporting field merchandisers where connectivity can be unreliable. **ML Kit barcode scanning** can read data encoded using most barcode formats; reliable on-device decoding with no network calls further supports offline-first.

Trade-offs: No cloud storage/functionality/sync in v0.1.0; offline-first doesn't support network calls for UPC details.

4. DATA DESIGN

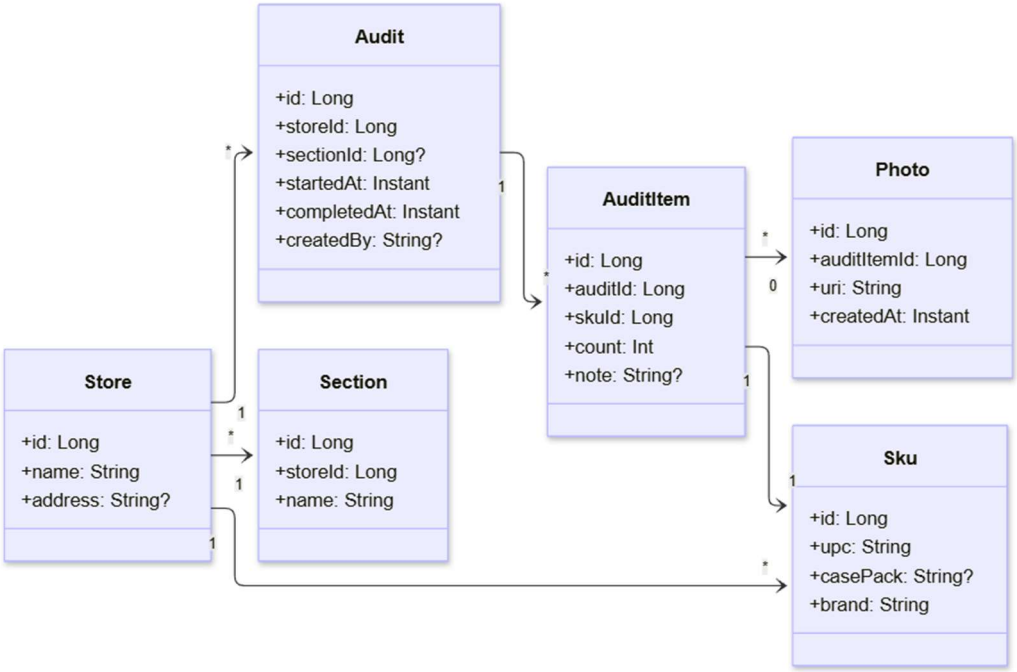
4.1 Data Description

Room is used for persistence. Photos are stored in app-private storage (or MediaStore) with URI references in the DB. Reports (PDF/CSV) are written to app-private files and optionally exported/shared. All timestamps are UTC (instant) with local display at UI.

4.2 Data Dictionary

Entity	*Attributes*	*Description*
Store	id: Long (PK), name: String, address: String?	Retail location
Section	id: Long (PK), storeId: Long (FK), name: String	Store area (e.g., aisle)
Sku	id: Long (PK), upc: String, name: String, casePack: String?, brand: String	Product master record
Audit	id: Long (PK), storeId: Long (FK), sectionId: Long? (FK), startedAt: Instant, completedAt: Instant, createdBy: String?	Single audit
AuditItem	id: Long (PK), auditId: Long (FK), skuId: Long (FK), count: Int, note: String?	Per-SKU audit item
Photo	id: Long (PK), auditItemId: Long (FK), uri: String, createdAt: Instant	Optional photo attached to audit item

Cardinality Rules



5. COMPONENT DESIGN

Key Interfaces:

```
// Key interfaces that use cases rely on
interface BarcodeScanner {
    function scan() return Result<String> // returns UPC details
}

interface SkuRepository {
    function getByUpc(upc: String)
        return Sku? // returns SKU
    function search(text: String)
        return List<Sku> // returns list of SKUs that match
    function add(sku: Sku)
        return Sku // adds SKU to catalog
}

interface AuditRepository {
    function openOrCreate(storeId: Id, sectionId?: Id)
```

```
        return Audit // opens current audit or starts new
    function addAuditItem(auditId: Id, skuId: Id, count: Int, note: String?)
        return AuditItem
    function attachPhoto(auditItemId: Id, fileUri: String)
        return Photo
    function getHistory()
        return List<Audit> // returns list of audits
}

interface ReportGenerator {
    function generatePdf(auditId: Id)
        return ReportFile // {uri, bytes?, path}
}
```

Core Operations:

```
// Check if audit already exists
AuditRepository.openOrCreate(storeId: Id, sectionId?: Id):

if exists audit where completedAt is null and
    storeId==storeId and sectionId==sectionId
    return audit
else:
    return insert Audit { storeId, sectionId, startedAt=now(), createdBy=String}


// Validate count and add audit item
AuditRepository.addAuditItem(auditId, skuId, count, note?):

    validate count >= 0
    SET item = AuditItem { auditId, skuId, count, note? }
    insert item
    return item

// Attach photo
AuditRepository.attachPhoto(auditId, uri):

    if not fileExists(uri): throw FileMissing
    insert Photo { auditItemId, uri, createdAt=now() }

// Scan SKU
SkuRepository.getByUpc(upc):

    return query Sku where upc == upc limit 1
```

```
// Search SKU
SkuRepository.search(query):

    return query Sku where name LIKE query or brand LIKE query
        order by name

// Add SKU
SkuRepository.add(sku):

    SET sku = Sku { skuId, upc, casePack?, brand }
    insert sku
    return sku

// Report generator
ReportGenerator.generatePdf(auditId):

    compose cover: store, section, timestamp
    for each AuditItem:
        render line: SKU (brand/name/size), count, note
        render line: upc
        if photos exist: append after upc
    write to file /reports/{auditId}-{timestamp}.generatePdf
    return uri

// Sharing
intent = ACTION_SEND_MULTIPLE
intent.putParcelableArrayListExtra(EXTRA_STREAM, [pdfUri])
intent.setType("application/pdf")
startActivity(chooser(intent))
```

6. HUMAN INTERFACE DESIGN

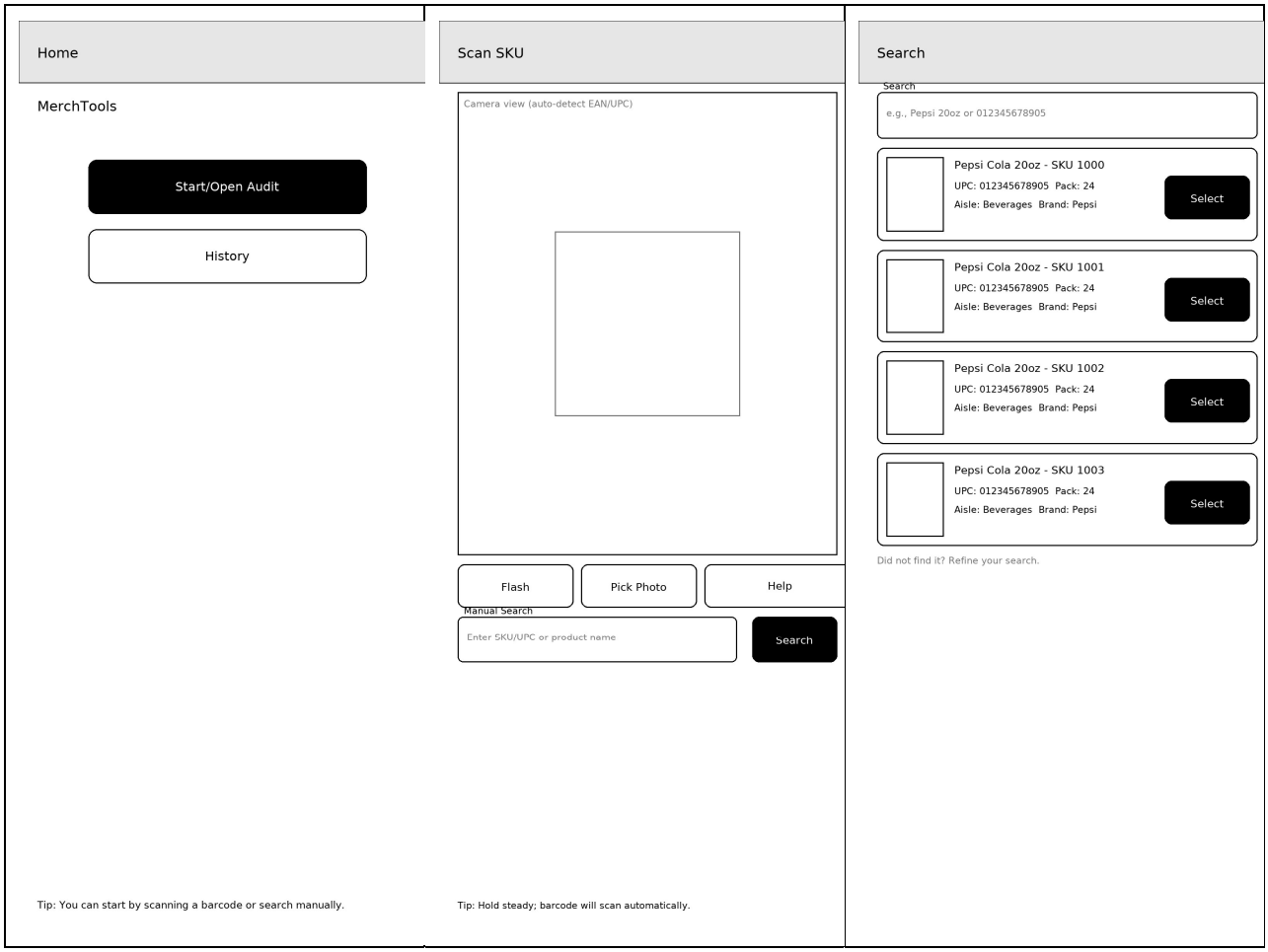
6.1 Overview of User Interface

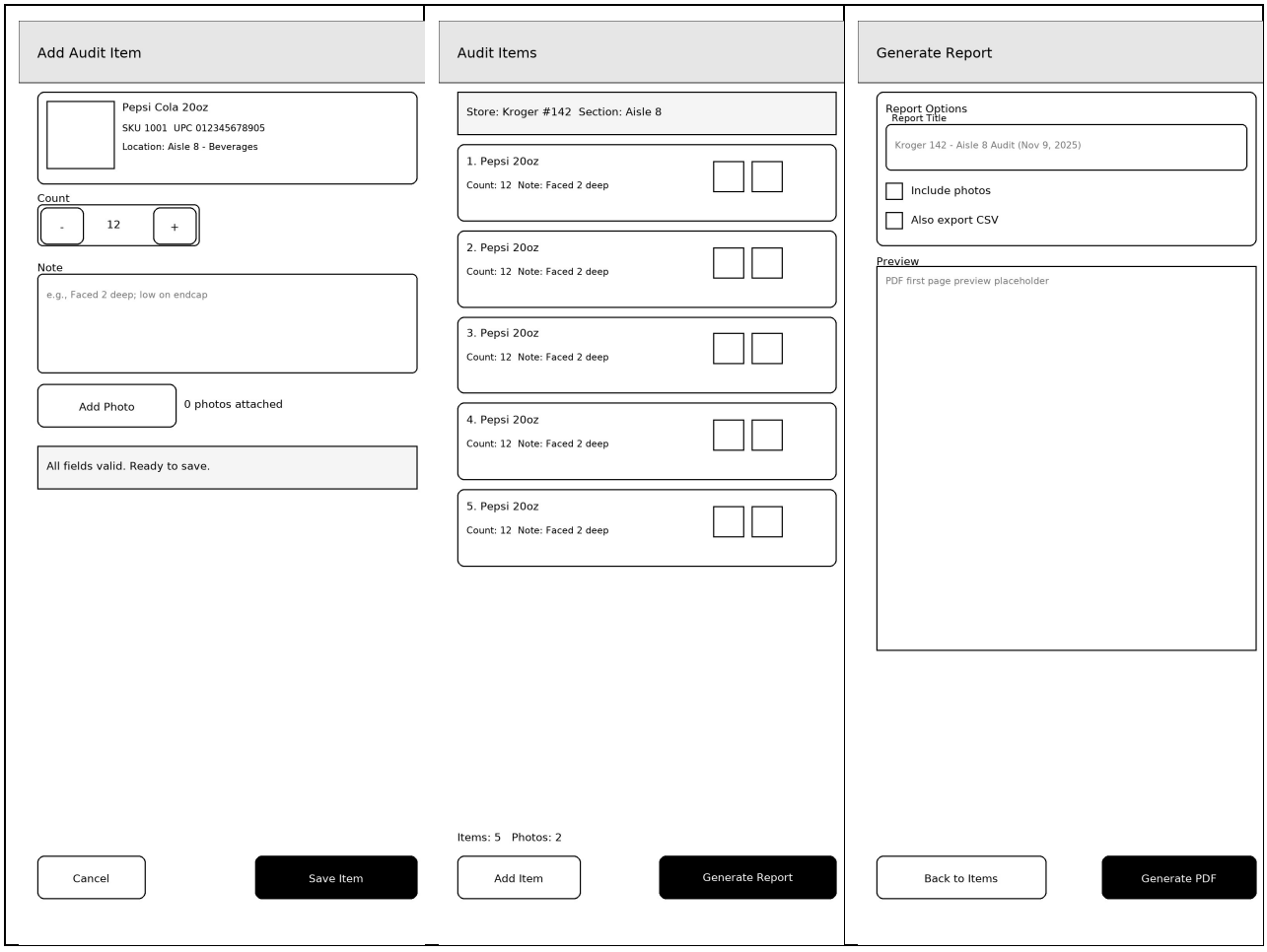
- Home: “Start/Open Audit” (select Store/Section) and “History”.
- Scan/Search: Big Scan button; text field for manual search.
- Add SKU: Add SKU button; displays blank SKU template with text fields; Scan button

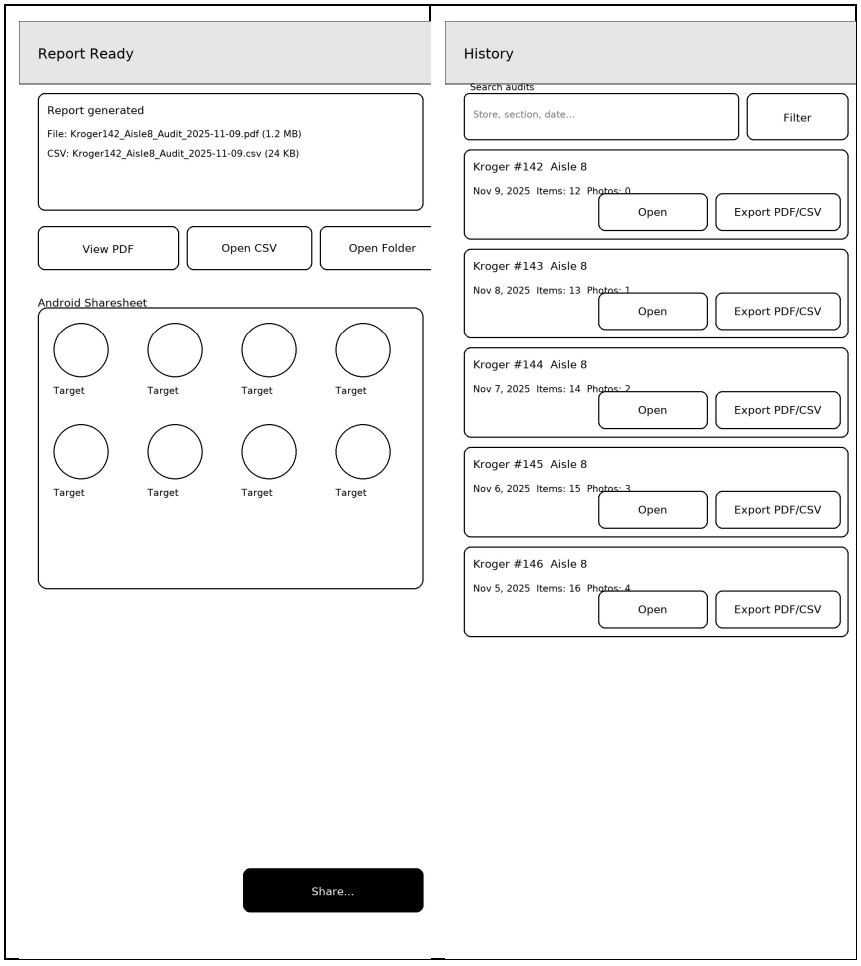
- Add Item: Shows selected SKU, count stepper, note field, “Add Photo” button, Save.
- Audit List (in-session): Running list of items with counts and photo; actions: edit/remove.
- Report: Summary preview, toggle “Generate”.
- Share: After generation, buttons: “Share Report” and “Open PDF”.
- History: List of past audits with date/store/section; open, share.

User feedback: toasts for saves, inline validation for counts, toast for PDF build, error dialogs for camera/permission issues.

6.2 Screen Images




Add Audit Item
Pepsi Cola 20oz
SKU 1001 UPC 012345678905
Location: Aisle 8 - Beverages
Count
-
12
+
Note
e.g., Faced 2 deep; low on endcap
Add Photo
0 photos attached
All fields valid. Ready to save.
Cancel
Save Item
Audit Items
Store: Kroger #142 Section: Aisle 8
1. Pepsi 20oz
Count: 12 Note: Faced 2 deep
2. Pepsi 20oz
Count: 12 Note: Faced 2 deep
3. Pepsi 20oz
Count: 12 Note: Faced 2 deep
4. Pepsi 20oz
Count: 12 Note: Faced 2 deep
5. Pepsi 20oz
Count: 12 Note: Faced 2 deep
Items: 5 Photos: 2
Add Item
Generate Report
Generate Report
Report Options
Report Title
Kroger 142 - Aisle 8 Audit (Nov 9, 2025)
Include photos
Also export CSV
Preview
PDF first page preview placeholder
Back to Items
Generate PDF



6.3 Screen Objects and Actions

- Buttons: Scan (opens camera), Add SKU (open Add SKU screen), Add Photo (invokes camera/gallery), Generate (builds artifacts), Share (invokes Sharesheet), History
- Inputs: Count (int ≥ 0, stepper & keyboard), Note (multiline, max N chars)
- Lists: Search results (select SKU), In-session items (tap to edit), History (tap to open)

7. REQUIREMENTS MATRIX

SRS Req	Use Case(s)	Components
FR-3.1 Scan/Search	UC-01, UC-02	BarcodeScanner, PermissionManager, SkuRepository
FR-3.2 Counts/Notes/Photos	UC-04	AuditRepository, AuditViewModel
FR-3.3 Generate PDF	UC-05	ReportGenerator, AuditRepository
FR-3.4 Share	UC-06	ShareService, ReportViewModel
FR-3.5 History	UC-07	AuditRepository, HistoryViewModel
FR-3.6 Add SKU	UC-03	SkuRepository, BarcodeScanner, PermissionManager

8. APPENDICES