Project Plan

Keeping Stock

CS 467/Winter/2026

Richard Phan, David Skaggs, Tran Trinh, Rich Liu

Change Log

Who did what and when. Helps to have an audit trail IRL. Table can grow as needed.

Change	Author	Date
Added to Introduction, Problem Statement, & Requirements	Richard Phan	01/15/2026
Add mock-up UI design	Tran Trinh	01/16/2026
Added Technologies and Important Resources.	Rich Liu	01/17/2026
Added design/architecture	David Skaggs	1/18/2026
Updated mock-up UI	Tran Trinh	01/18/2026
Updated design/architecture	David Skaggs	01/19/2026
Added to my development plan	Richard Phan	01/19/2026
Added development plan	Rich Liu	01/19/2026
Added development plan	Tran Trinh	01/19/2026
Added Development Plan	David Skaggs	01/19/2026
Added Basic UX flowchart	David Skaggs	01/19/2026
Polished for final submission	Tran Trinh David Skaggs	01/20/2026

Introduction

Be brief. Aim for 2-3 paragraphs.
Try to highlight what makes your project unique and different

As individuals accumulate belongings at home or in office environments, it becomes increasingly difficult to keep track of what items they own and where those items are stored. Items are often placed into boxes, closets, drawers, or other storage spaces with minimal documentation, making retrieval time-consuming and frustrating. This lack of organization is a common issue that worsens as items are moved or rearranged over time.

Keeping Stock is a mobile inventory tracking application designed to help users catalog their belongings and associate them with specific storage locations. The application allows users to record items using photos, descriptions, and tags, making it easier to visually identify and search for stored items.

The distinguishing feature of Keeping Stock is its integration of physical storage spaces with digital records. By generating QR code labels for storage locations, users can scan a code and immediately view a list of items stored in that space. This approach creates a simple and practical system for maintaining an up-to-date inventory without requiring complex workflows or excessive manual input.

Problem Statement

The paper provides a concise description of the main problem that your project intends to solve. The description is easy to follow, mostly non-technical and high-level, and provides background where needed.

This section should closely follow the bullet format presented in "Exploration - How to Write a Problem Statement":

- ***State** the context of the problem (Context)*
- ***What** is the issue that needs to be solved? (Issue)*
- ***Who** has the problem or need? **Why** do they have this need? (Relevance)*
- ***How** may the issue be solved? (Objective)*

Single page at most. Aim for 4-5 paragraphs.

People commonly store personal or work-related items across multiple physical locations, such as boxes, shelves, closets, cabinets, and storage rooms. As time passes, these storage spaces are frequently reused, reorganized, or relocated, especially during cleaning, office restructuring, or moves. When items are stored without consistent documentation, users gradually lose track of what they own and where those items are placed. This issue becomes more noticeable for items that are accessed infrequently.

The primary issue is the absence of an efficient and user-friendly system for tracking stored items and their locations. Common approaches, such as handwritten labels, basic spreadsheets, or relying on memory, are prone to error and are difficult to keep up to date. Labels may become unreadable or detached, spreadsheets require manual updates that users often neglect, and memory-based tracking becomes unreliable as the number of stored items increases. As a result, these methods fail to provide a dependable long-term solution.

This problem affects a wide range of users, including homeowners, renters, students, and office workers who need an organized way to manage their belongings. In both personal and professional environments, the inability to quickly locate items leads to wasted time and unnecessary frustration. Users may also forget what items they already own, leading to duplicate purchases or inefficient use of available storage space. Over time, this lack of organization can negatively impact productivity.

The objective of Keeping Stock is to provide a simple and accessible mobile application that allows users to record items, associate them with specific storage spaces, and retrieve that information quickly when needed. By combining item photos, searchable descriptions, editable tags, and QR code-based identification of storage locations, the system aims to create a centralized and reliable inventory solution. This approach reduces the effort required to maintain accurate records while making it easy for users to locate stored items efficiently.

Requirements

A brief introduction to the project requirements. This should logically follow your problem statement. 1-2 paragraphs.

*A bullet point listing of the **most important** Requirements/Stories/Features/Tasks for your software or system. Generalize as much as possible. This should be high-level and not exhaustive.*

Aim for 1-2 pages.

Keeping Stock is designed to meet the needs outlined in the problem statement by providing a centralized and user-friendly inventory tracking system. The application focuses on helping users visually document their belongings, associate items with physical storage locations, and retrieve that information efficiently when needed. Emphasis is placed on simplicity, accuracy, and ease of maintenance to ensure the system remains practical for everyday use.

The system prioritizes visual identification through photos and a gallery-style interface, reducing the need for extensive manual data entry. By linking items to clearly defined storage spaces and enabling quick retrieval through search and QR code scanning, the application minimizes the effort required to manage stored belongings. All requirements are intentionally scoped to ensure the project is achievable within the timeline.

The following are the high-level functional and non-functional requirements, which are representative examples to illustrate key system behaviors for our proposed app. It is not exhaustive and serves as an early design guide:

- Users can add items by uploading or taking photos to create an item card
  <u>Representative Functional requirement:</u> CameraX captures a still image.
  <u>Representative Non-Functional requirement:</u> Camera capture will have a reasonable latency on a reference pixel phone and within storage constraints.
- The system stores item descriptions and editable tags
  <u>Representative Functional requirement</u>: Users are able to enter a description and add characters to tag each item. The Tags could also be optionally input using AI image recognition.

    Representative Non-Functional requirement: Tag search index should update within a reasonable time, even if the item list grows on the reference pixel phone.
- Users can create and manage storage containers (boxes, closets, cabinets, etc)
  Representative Functional requirement: The "Create Space" screen should allow the entry of characters for the name.
  Representative Non-Functional requirement: The space list scroll should have reasonable performance when rendering the spaces.
- Each storage space is assigned a unique, scannable QR code
  Representative Functional requirement: On space creation, the system will generate a scannable QR code.
  Representative Non-Functional requirements: QR decoding shall compute within a reasonable time under ambient light conditions.
- Scanning a QR code displays a list of items stored in that space
  Representative Functional requirement: After a successful scan, the app shall open the space in the app.
  Representative Non-Functional requirements: Incorrect or ambiguous scans will be minimized.
- Users can search for items using keywords and tags
  Representative Functional requirement: Search bar accepts substrings greater than a couple of characters and returns matching items.
  Representative Non-Functional requirement: Search results should appear within a reasonable time to support an interactive use case.
- Items can be marked as "stored" or "taken out" to reflect their current status
  Representative Functional requirement: A toggle or button on the item card that will change the state of the object as stored or taken out.
  Representative Non-Functional requirement: Changes to an item state are intended to update persistently and reflect without a noticeable delay.
- Items are displayed in a gallery-style visual interface
  Representative Functional requirement: The gallery will show in a grid view.
  Representative Non-Functional requirement: Memory usage shall be reasonable to avoid excessive resource use on the device when displaying larger collections.

Stretch Goal (If time permits):
- The application may support customized visual themes, such as dark mode, to improve usability and accessibility in different lighting conditions.

Design / Architecture

A description of your initial thoughts on the structure of the software and/or system. This should be high-level only and not exhaustive, and should logically follow from your Requirements.

- *First, break the system down into high-level functional areas.*

- *Provide a brief discussion of how each of these areas will support the listed Major Requirements from above. You do not need to be detailed with this. Just get enough information across so that a technical reader will understand your ideas.*
- *Finally, iterate on this a bit with the team. All team members should have a chance to review.*
- *Include **at least two** graphical examples that demonstrate a screen or page mockup, design, architecture, flowchart, layout, etc. Choose parts of your Design or Architecture or UI/UX that would benefit from a visual explanation. These should be detailed and informative. Captions required.*

Aim for 2-4 pages. You should use bullet lists, flow charts, block diagrams, screen-mockups, etc, to help portray your high-level design.

The functional areas of the app include: UI/Presentation, item management, storage space management, media capture, QR code generation, search and retrieval, and data persistence (DB). These areas do not map directly or strictly to architectural layers. Each functional area spans multiple layers of the application architecture, including the user interface, state management, and data persistence layers.

The functional area for the UI/Presentation will be responsible for presenting information to the user in a clear manner. It will provide a gallery view for browsing items, a list and/or hierarchical view of the storage spaces/containers for navigation between them, and detail screens for viewing and editing individual items or containers/spaces. This layer directly supports the requirements related to visual identification, ease of navigation, and low effort interaction by emphasizing the use of photos, a grid-based layout, and simple controls in the form of buttons for common actions such as adding items and spaces, scanning QR codes, updating item status, deleting items and spaces, and other tasks.

The functional area related to item management handles the creation, modification, and organization of individual inventory items. It will support the user adding a new item to a space, providing the item with a universal unique ID (UUID), storing item photos and descriptions, adding tags to items, and setting status indicators such as “stored” and “check-out”. By associating items with storage spaces and maintaining editable metadata related to the item, this area fulfills requirements related to documenting belongings, tracking their current state, and enabling accurate retrieval through search and browsing. Item management logic ensures that updates persist reliably and remain responsive.

The functional area for storage space management is responsible for creating and maintaining the representations of container/space structures, such as rooms, cabinets, closets, shelves, boxes, or anything else that may resemble or take on the classification of a “container” or a “space”. Each space maintains descriptive metadata related to the container, such as a UUID, container type, a pointer to the parent container (either pointing to null or the root parent container, e.g. “home” or “office”), and associations to contained items. This area supports requirements for organizing items by location and for efficiently displaying items within a selected space. The design allows the interface to scale as the number of spaces increases, while keeping navigation intuitive by maintaining a hierarchical container structure.

The media capture functional area manages photo creation/selection and image handling for items and containers. It integrates the device's camera or gallery functionality to allow users to select existing photos on the device or take photos directly within the app, as well as manage stored image assets for gallery display. This functional area supports the emphasis on visual identification by minimizing manual data entry and ensuring images load efficiently without excessive memory usage, even when displaying larger collections of items and containers.

The QR code generation and scanning functional area handles generating unique and scannable QR codes for storage containers and decoding scanned codes at runtime to automatically navigate to that container's space page. On space creation, this area produces a QR code that can be attached to the container/storage location. This directly supports requirements for fast, low-effort retrieval of stored items while ensuring scans remain reliable and responsive under typical usage conditions.

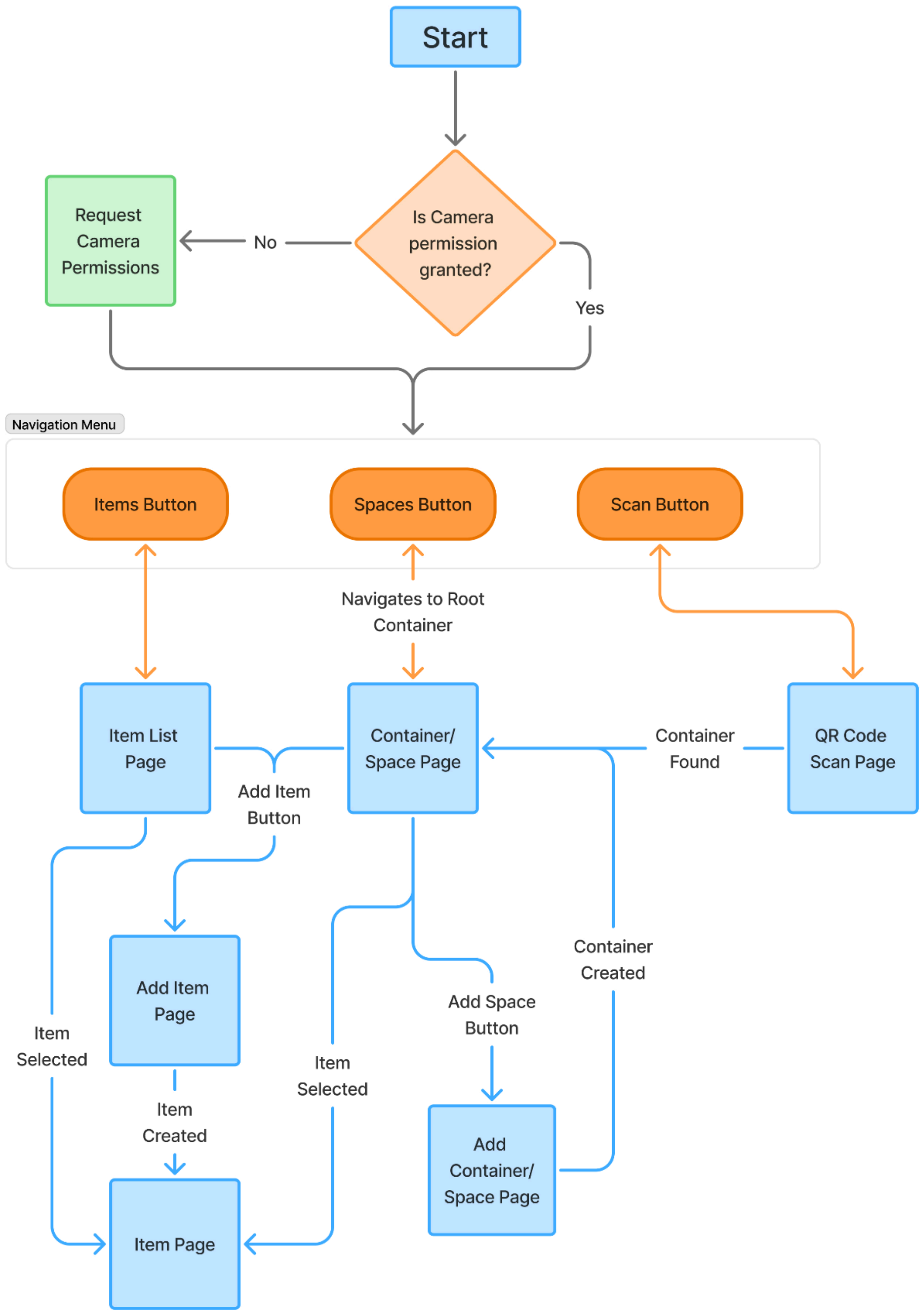
The search and retrieval functional area enables users to quickly locate items using keywords, tags, and partial text matching. It provides responsive search behavior for easy interactive use, even as the dataset grows by use of indexing item metadata and coordinating with item and space associations.

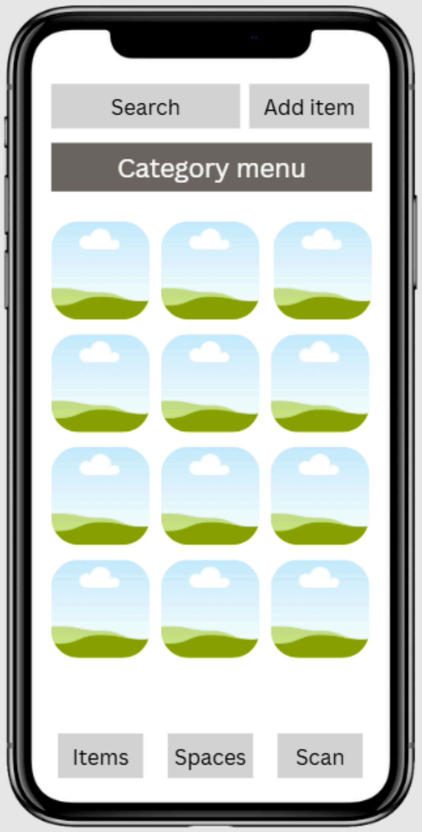
The data persistence and state management functional area ensures that all items, containers, images, tags, etc are stored reliably on the device (we will be using local device storage) and restored consistently across sessions. It supports non-functional requirements related to performance, responsiveness, and data integrity by managing structured storage and update propagation throughout the system.

At the high level, the application follows an MVVM architecture, with Jetpack Compose acting as the View, ViewModels managing UI state and user-driven logic, and repositories and data sources forming the Model layer.

- First, we have the View layer, which shows the pages of the app and lists, images, buttons, and the display of items and containers in a grid-based style. It collects user input and displays the current State.
- Next, we have the ViewModel layer, which deals with State management and Behavior of the app. It holds the screen state, validates input, coordinates actions, and calls the repository layer.
- The repository layer comes next, and it is the layer that talks to the database, acting as the middleman between the logic and storage. It reads/writes to the DB.
- Then we have the database/storage layer itself, which stores items, containers, tags, and image URIs.
- Finally, we have the platform services integrations, which isn't quite a layer as it is used by the View and ViewModel layers. These integrations are for QR generation and scanning, using the device's camera, and AI image tagging.

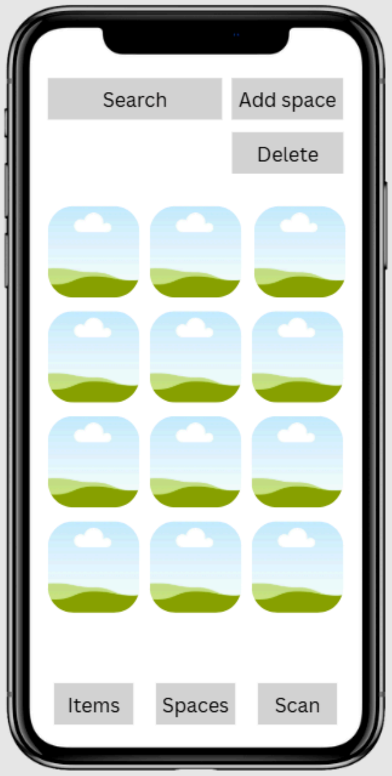
Basic UX Flowchart





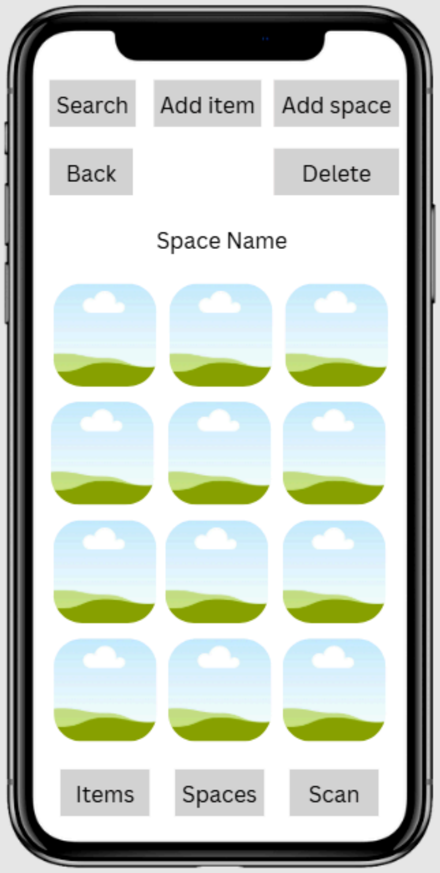
Items List Page

- Show a list of items from all the spaces.
- Have a search bar to search for items.
- Have a menu to sort the items into categories.
- Have an add button to add items.



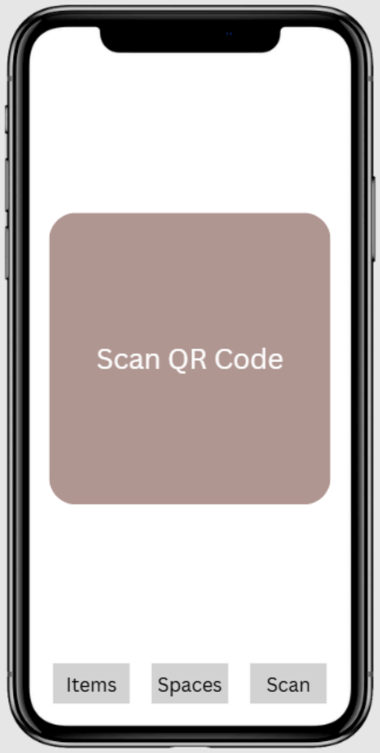
Spaces List Page

- When the user clicks on the “Spaces” button at the bottom of the screen, the app will show a list of spaces that do not have parents.
- Have a search bar to search for a specific space.
- Have an add button to add a space.
- Have a delete button to delete a space. User is unable to delete the space if it still contains sub-spaces or items.



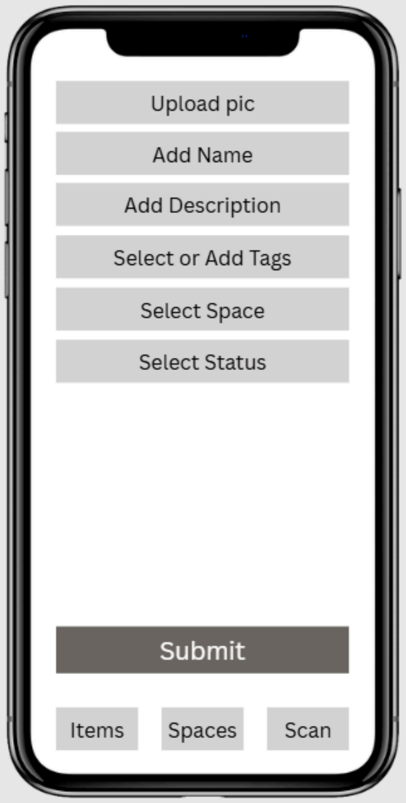
SubSpaces List Page

- When the user selects a space, the app will show the space's name, a list of sub-spaces and/or items that belong to that space.
- Have a search bar to search for a specific space or item.
- Have an add button to add an item or a space.
- Have a delete button to delete a space. User is unable to delete the space if it's not empty - Delete will have options such as: delete all sub-spaces, delete all items, or delete all. with pop-up warnings after selection. The deletion rule above still applies. Once a user selects delete all and there are sub-spaces that are not empty, the app will only delete the empty spaces and give a notice that the other non-empty spaces cannot be deleted.
- Have a back button to go back to the space's parent.



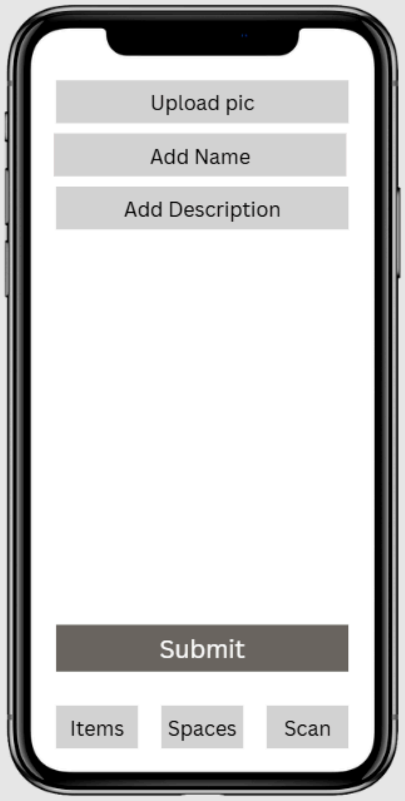
QR Code Scan Page

- Scan the QR code on the space
- Will lead to a list of items within that space.



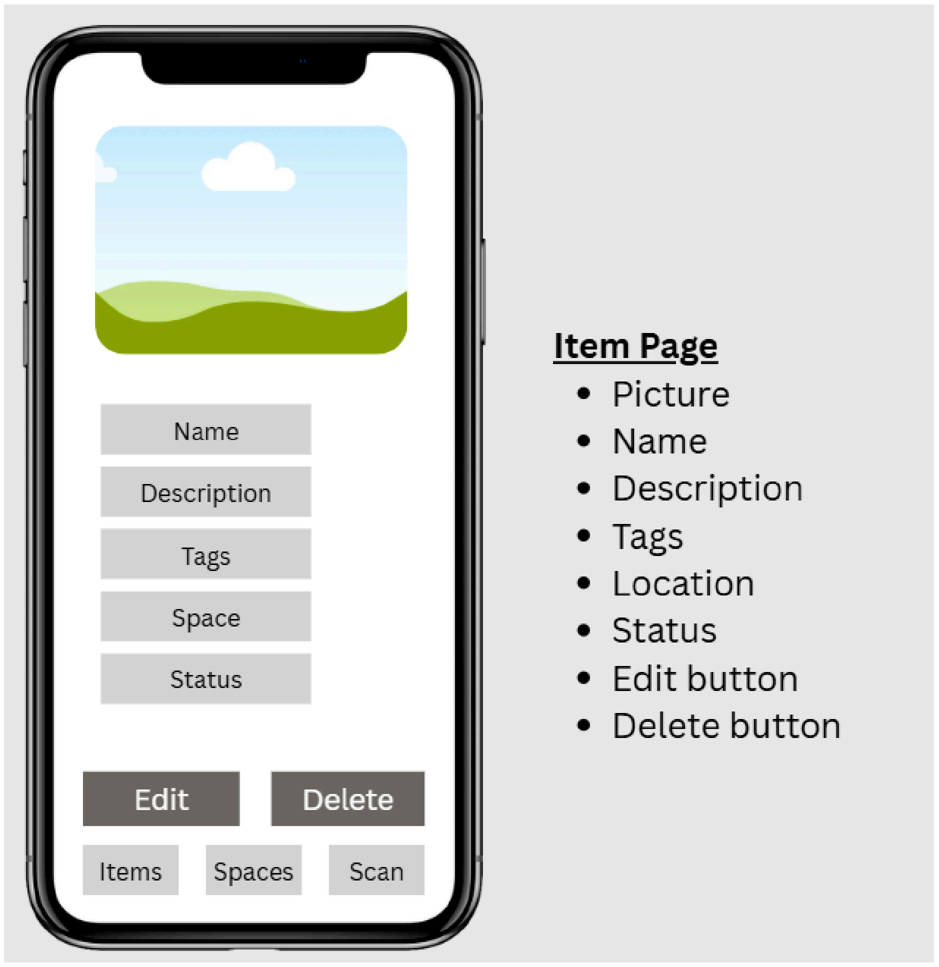
Add Item Page

- Prompt to add Picture, Name, Description, Tags, Location, Status.
- Tags will be used for the category menu.



Add Space Page

- Prompt to add Picture, Name, Description.
- Will generate a QR code associated with the space after submit.



Technologies

Include a bullet point listing of the ***most important*** *software libraries, languages, APIs, development tools, servers, or other systems that you expect to use.*

- Kotlin - Main programming language for Android
- Jetpack Compose - UI toolkit for building the frontend for Android
- Room - Android wrapper library for SQLite
- CameraX - Camera APIs for Camera and Scanning QR codes
- ML Kit - QR Code Scanning and Image Labeling APIs
- Android Studio - Official Android IDE with emulator and debugging tools
- GitHub - For version control
- Gradle - For dependency management and build automation

Include a brief discussion (1-2 paragraphs) on one important library, technology, or language that you chose. ***Take into account the pros and cons of the technology and especially the relative strengths and weaknesses in your team.***

Android with Koltin and Jetpack Compose was selected as the main language and toolkit for this project due to the fact that it has tight integration with modern Android development and Architecture, and has become the primary and preferred language for developing Android Applications. Compose allows the UI to react directly to State changes, which aligns well with our planned MVVM architecture and reduces the amount of additional overhead typically required when using something like XML layouts instead. While Compose is newer than the traditional XML-based approach, having been officially supported starting in 2017 and Google outright stating it is the preferred choice around 2019 [1], it is now the recommended UI toolkit for Android and is supported by extensive official documentation within Android Studio.

The advantage of Kotlin/Jetpack Compose versus the older Java/XML approach is that the UI code should be more intuitive because it is declarative, so rather than dealing with separate XML screens and linking buttons with IDs, one can type a shortlist and it displays it on the screen. Another advantage is fewer lines of code compared to Java, easier to read and understand. The reduction in boilerplate code and the enhanced support that this modern approach provides as opposed to the XML-based approach is expected to benefit the project in the long-run.

From a team perspective, Kotlin + Compose presents several advantages as well as some major risks. Some team members have prior experience with a Kotlin and Compose-based UI development approach, which allows early progress on screen layout and navigation. Other members are less familiar with these technologies, which will introduce a learning curve for these teammates in addition to their own responsibilities and tasks. To mitigate this risk, UI logic will be centralized in shared composables, previews will be used to allow isolated UI development, and ViewModels will expose clearly defined UI state objects to reduce complexity in the View layer. These tasks are assigned to the teammates with the most experience with these technologies, while the other teammates bring themselves up to speed with the relevant technologies as well as complete their own initial tasks.

Important Resources (if any)

Include a bullet point listing of any external resources you expect to use. Include things like external web resources, training datasets, graphics, systems, hardware, etc.

- Android API Reference - https://developer.android.com/reference
- Android Studio - https://developer.android.com/studio
- CameraX Reference - https://developer.android.com/media/camera/camerax
- Material Components - https://developer.android.com/design/ui/mobile/guides/components/material-overview
- Room SQLite wrapper - https://developer.android.com/training/data-storage/room

Development Plan

*This is a listing of the **high-level project tasks** for EACH Team member, complete with a time estimate in hours, for each of the major project milestones. Make sure to include tasks and time for setting up your development environment and learning new technologies.*

***Important:** The amount of time spent per Team member must be roughly equivalent, and should be 10 hours per week minimum. If teammates do not have enough to do, the project must be expanded (talk to your PM, include stretch goals, etc). If the work seems like too much, we will have you modify the plan.*

- **Team Member David Skaggs (UI Lead)**
- **Progress Report #1**
    - Set up Android development environment, project structure, and Jetpack Compose basics.
        - **Time estimate:** 6 hours
        - **Dependencies/Resources/Risks:** Refamiliarizing myself with Jetpack Compose navigation patterns
    - Refine UI wireframe and determine navigation flow
        - **Time estimate:** 2 hours
        - **Dependencies/Resources/Risks:** Need UX clarity on unified tag selection overlay component. The navigation flow will be discussed with the integration lead
    - Implement base navigation graph and placeholder screens based on UI wireframe
        - **Time estimate:** 8 hours
        - **Dependencies/Resources/Risks:** Coordinate with the integration lead for the implementation of the base navigation graph and hooking up placeholder screens.
- **Progress Report #2**
    - Implement the Item List page with search UI and item cards
        - **Time estimate:** 8 hours
        - **Dependencies/Resources/Risks:** Must discuss with Database and Repository lead
    - Implement Container view page displaying nested containers and item lists within that container.
        - **Time estimate:** 8 hours
        - **Dependencies/Resources/Risks:** Coordination with QR and integration Leads regarding clarification on the implementation of QR creation/scanning process, as well as QR scan UI.
    - Implement Item view page
        - **Time estimate:** 4 hours
        - **Dependencies/Resources/Risks:**
- **Progress Report #3**

- Implement Add/Edit Item and Add/Edit Container Pages/Forms
  - **Time estimate:** 12 hours
  - **Dependencies/Resources/Risks:** The tag selection overlay component may require assistance from other team members to properly implement. Integration with QR and validation logic of adding/editing items and spaces will need coordination with the database and the integration lead.
- Polish UI, review empty states, loading indicators, permissions requests, and error states
  - **Time estimate:** 10 hours
  - **Dependencies/Resources/Risks:** Time estimate is VERY dependent on prior progress and existing issues between other team leads, as well as any possible pain points encountered during development.

- **Project Archive - Final**
  - Coordinate with the integration lead for full integration testing across all layers and team leads. Fix bugs, resolve edge cases, improve reliability and performance, polish UI.
    - Time estimate in hours: 10
    - Major Dependencies/Resources/Risks (if any)
- **Total Hours: 68 hours**

- **Richard Phan**
- **Progress Report #1**
  - **High-level Requirement/Story/Feature/Task** - Set up the Android development environment and implement the core camera functionality using CameraX, including requesting camera permissions, capturing photos of items, and storing image files locally so they can be referenced throughout the application.
    - **Time estimate in hours** - 15 hours
    - **Major Dependencies/Resources/Risks (if any)**
      - Requires learning CameraX APIs and Android permission handling
      - Risks include inconsistent camera behavior across devices and initial setup challenges in Android Studio.
- **Progress Report #2**
  - **High-level Requirement/Story/Feature/Task** - Integrate ML Kit image labeling to analyze captured item photos and automatically generate a preliminary item description and suggested tags, then pass this information to the app's ViewModel layer so users can review and edit the results.
    - **Time estimate in hours** - 15 hours
    - **Major Dependencies/Resources/Risks (if any)**
      - Depends on successful image capture and storage
      - AI-generated labels may be vague or inaccurate, requiring careful handling and user-editable output.
- **Progress Report #3**

        - **High-level Requirement/Story/Feature/Task** - Implement QR code functionality by generating unique QR codes for storage locations and enabling QR code scanning so users can quickly retrieve a list of items stored in a scanned location.
            - **Time estimate in hours** - 12 hours
            - **Major Dependencies/Resources/Risks (if any)**
                - Camera resource conflicts between photo capture and QR scanning
                - QR scan reliability may vary depending on camera focus and device-specific camera behavior.
- **Project Archive - Final**
    - **High-level Requirement/Story/Feature/Task** - Complete full integration testing of camera capture, AI-based tagging, and QR code features, resolve bugs discovered during integration with the UI and database layers, and document the implemented functionality for the final project submission.
        - **Time estimate in hours** - 8 hours
        - **Major Dependencies/Resources/Risks (if any)**
            - Late-stage integration issues with other components
            - Limited availability of multiple physical devices for thorough testing.
- **Total Hours: 50 hours**

- **Team Member Name: Tran Trinh**
- **Progress Report #1**
    - High-level Requirement/Story/Feature/Task -
        - Learn Room database concepts and design the initial data models for items and spaces, including basic relationships and fields.
    - **Time estimate in hours: 12 hours**
    - **Major Dependencies/Resources/Risks**
        - Learning new tools and data models decisions may impact teammates' features.
- **Progress Report #2**
    - **High-level Requirement/Story/Feature/Task -**
        - Implement Room, configure the database, and build a data access layer that the rest of the app can use.
    - **Time estimate in hours: 14 hours**
    - **Major Dependencies/Resources/Risks**
        - Room configuration errors and difficulties of changing the database structure once the app implementation is in progress.
- **Progress Report #3**
    - **High-level Requirement/Story/Feature/Task -**
        - Implement query logic for searching and filtering by tags, spaces, and item status.
    - **Time estimate in hours: 14 hours**

        - **Major Dependencies/Resources/Risks**
            - Query complexity and ensuring performance with increasing datasets.
- **Project Archive - Final**
    - **High-level Requirement/Story/Feature/Task -**
            - Ensure data correctness across the app and integration testing.
        - **Time estimate in hours: 10 hours**
        - **Major Dependencies/Resources/Risks**
            - Discovering edge cases during final testing and integration.
- **Total Hours: 50 hours**

- **Team Member Name: Rich**
- **Progress Report #1**
    - **High-level Requirement/Story/Feature/Task -** Set up the Android development environment, establish the MVVM project architecture, configure navigation, and create base ViewModels and interfaces to connect the UI, data layer, and platform services.
        - **Time estimate in hours -** 14 hours.
        - **Major Dependencies/Resources/Risks (if any):**
            - Requires getting acclimated to the stack (Android Studio, MVVM architecture, and Jetpack Compose).
            - Dependent on coordination with UI/Data/Platform leads.
            - Risk of refactoring early architectural decisions.
- **Progress Report #2**
    - **High-level Requirement/Story/Feature/Task –** Implement feature wiring between UI, ViewModels, Repositories, and platform services. This includes the request/response model and handling basic app flows.
        - **Time estimate in hours -** 14 hours.
        - **Major Dependencies/Resources/Risks (if any)**
            - Dependent on repository interfaces and data schema from the data lead.
            - Dependent on CameraX, QR Scanner, and AI tagging from Platform Lead.
            - Risk of interoperability between data formats or interfaces.
- **Progress Report #3**
    - **High-level Requirement/Story/Feature/Task –** Integrate workflows end-to-end, replace mocks with real integration, and implement error handling. Resolve any interface mismatch between the differing pieces of software and manage states across the layers.
        - **Time estimate in hours –** 12 hours.
        - **Major Dependencies/Resources/Risks (if any)**
            - Depending on the completion of CameraX, QR scanner, and AI tagging features from the Platform lead.
            - It depends on a stable UI flow from the UI lead.

      - Risk of life cycle issues, race conditions, and states not performing consistently.
- **Project Archive - Final**
  - **High-level Requirement/Story/Feature/Task –** Perform full integration testing across UI, data, and platform services. Fix bugs, resolve edge cases, improve reliability and performance.
    - **Time estimate in hours –** 10 hours.
    - **Major Dependencies/Resources/Risks (if any)**
      - Late-stage integration issues across features.
      - Limited testing time.
      - Risk of discovering issues late in development.
- **Total Hours:** 50 hours.

Conclusion

Me realizing I could've just checked Keeping Stock instead of buying a 4th pair of scissors
imgflip.com

References

If needed. Please use the following style:

This is the body of your paper here. Text text text, blah blah blah [1]. Some more text.

(and below in References)

[1] Wikipedia contributors. (n.d.). Kotlin. In Wikipedia, The Free Encyclopedia. Retrieved January 21, 2026, from https://en.wikipedia.org/wiki/Kotlin