# Main prototype requirements

1. Register items into the stock take app

1. Via stock take managment page on the web app

2. Via stock take app end user interface (self service interface via touchscreen near the actual stock)

2. Each item in the database must have the following properties

1. Id

2. (Object) ItemTag

3. (Object) History

4. (Object) Location

5. (Optional) Description

6. (Optional) MF number

7. (Optional) SS number

8. (Optional) Condition

9. (Optional) Configured

3. ItemTag objects, that group items by a selected property to allow quick queries covering large number of items

1. Id

2. Item name

3. (Enum) Category

4. TrackedItem

5. (Optional) Description

6. (Optional)(Enum) Subcategory

ItemRecord

(you could technically have all of these properties inside ItemTag, but then the ItemTag class would have to hold the definition of the item it represents AND keeping track of its current state. ItemRecord keeps track of the actual state of the items)

1. Id

2. Item

3. Qty

4. Locations

5. Qty\_threshold (to be used for low stock warnings)

6. Pinned (as important)

History

(holds CRUD operations details perfomed for each item)

1. Id

2. CRUD operation details

3. Date

4. Agent

Location

1. Id

2. Description

BulkOperation - keeps records of all bulk operations. This could be used to track how much new equipment has been put into the stock

1. Newstock - determine whether items are reused or whether it’s new items

2. (Optional)Description

Orders - represents equipment that has been ordered, but not yet added to the stock. This will save admins and operatives from raising tickets for low stock that is already being taken care of.

Category -

Design notes

1. Should order be part of ItemRecord or a seperate entity? Orders can be comprised of multiple item types, which would make them icomptabile with ItemRecord that are dedicated to a single tpye of item. However, I WAS cautious about ItemRecord depending an external entity(Orders) for its pinned state change, which could lead to hard to detect bugs, as it may be difficult(once the codebase expands) to determine all external dependencies of ItemRecord with such loose coupling. But that’s a risk that I believe that is worth taking, as I can see the “pinned” property of ItemRecord being used by other parts of the app. Perhaps implement the “pinned” property as a service / seperate entity.

**Stock take app required functionality**

1. Self service interface - searching for items

**1.1 Searching for items with interactive mode**

a. List item categories

b. List item sub-categories, if defined

c. List all items within selected category

**1.2 Searching items via searchbox**

a. Users can find their items by typing into a search box, which lists all matching items, categories, and sub-categories

**1.3 Searching for an item via one of its properties**

a. Users can use SS / MF number to identify the item in the app.

2. CRUD operation types

**2.1 Tracked items**

a. Bulk operations are unavailable, each item must be scanned off individually

b. It is possible to configure a CRUD operation type that defines the desired CRUD operation, which can be ap plied to items provided by user (similar to how quick edit allows to lock in a custom operation)

**2.1 Untracked items**

a. CRUD operations only require the item id and qty, thus can be performed on bulk item quantities

**The app must support several different types of workflows when performing CRUD operations against the stock DB:**

1 On demand stock requests for day to day operations

1.1 Users will prioritize speed of CRUD operations

1.2 CRUD operations are likely to be simple, allowing for simple and limited UI optimized for content visibil ity and ease of operation, meaning the controls to perform CRUD operations must be simple

1.3 Accuracy, input values and UI controls must not allow ambigious touchscreen input to avoid needless UI prompts and inaccurate input values.

2 Large stock operations, usually completed by stock admins

2.1 Users will priroritize adaptability of CRUD operations

2.2 The UI should allow to configure various CRUD operations and workflows:

- Uploading items in batches

- Overriding default values on items, such as “Configured”, “Condition”, etc

- Better visiblity for CRUD operations that require data to be input in succession.

3 Admin stock view(can be refactored at a later date for users needs)

3.1 Not as crucial this early in the development lifecycle, as it will simply collate data from the DB and dis play it.

3.2 Capability to view stock levels, visibility on CRUD operation history.

3.3 Visual designs will allow for more efficient way of defining minimum requirements for view page

**On demand stock requests for day to day operations (Self service interface)**

1. Searching for items

**1.1 Searching for items with interactive mode**

a. List item categories as icons on the screen

b. List item sub-categories, if defined

c. List all items within selected category

**1.2 Searching items via searchbox**

a. Interface as above

b. Users can find their items by typing into a search box, which lists all matching items, categories, and sub-categories

**1.3 Searching for an item using one of its properties**

a. Interface as above

b. Users can use SS / MF number to identify the item in the app.

2. Performing Quick Edit CRUD operations on selected items

**2.1 Untracked items**

a. Items / item categories / item subcategories are listed as icons on the screen

b. Tapping on an item expands it on the screen, revealing UI elements, just below selected item, that allows the user to define their desired CRUD operation, i.e. select item amount, destination location, etc.

c. If the icon of the item is pressed again, the icon is minimized back to its original size

**2.2 Tracked items**

a. Items / item categories / item subcategories are listed as icons on the screen

b. Tapping on an item expands it on the screen, revealing UI elements, just below the selected item, that allows the user to define their desired CRUD operation: input box for a barcode, location dropbox, etc

**Large stock operations, usually completed by stock admins**

1. Bulk CRUD operations

1.1 Form / page dedicated entirely for bulk CRUD operation on stock

1.2 Provides clear visiblity on CRUD operation’s values, such as item type, item codition, location from and to, and others.

1.3 Only one type of item can be selected on when doing bulk CRUD operations

1.4 All details of the CRUD operation must be tracked and displayed (as the items are scanned to the page)

1.5 Review page before submitting desired CRUD operation

**Admin stock view(can be refactored at a later date for users needs)**

1. High level stock view (item records)

1.1 Users will prioritize high level view of current stock levels.

1.2 Users will appreciate visiblity on notable system events, such as low stock, late orders, unusually high item quantities specified in a CRUD request

1.3 Item records to be listed in a table with key information such as item name, category, location(s), qty, etc

1.4 The stock view table must be able to be sorted through using either of its columns

1.5 Users can search for an item record by entering a text query that will be compared to either of the table’s values

2. Stock view (item record details)

2.1 Item records collate items of the same type

2.2 Item records can be expanded, on a dedicated page, that displays all properties of the record, as well as list ing individual items linked to the item record

2.3 Individual items will be listed in a table with key information such as item name, category, location(s), qty, etc

2.4 The table must be able to be sorted through using either of its columns

2.5 Users can search for an item by entering a text query that will be compared to either of the table’s values

Mention how to specify location, condition, etc for untracked items!

New items! Let users create new items on the spot

Revamp on categories and subcategories. Categories can be nested as many times as desired now. And instead of each item having a category number, items are linked to Category objects via CategoryId