Welcome to your

Stockpile

Project proposal by Michael Lundie

GitHub Username: **michael-lundie**
www: **michael.lundie.io**
m: **michael@lundie.io**

Contents

GitHub Username: michael-lundie

Stockpile

Description

Achieve the peace of mind that comes with building a Stockpile for emergencies and natural disasters. Manage your emergency stockpile in a simple fashion. Get assistance creating a new emergency stockpile, create monthly stock-up goals and shopping lists and get alerted to expiring items.

Intended User

This application is intended for users living in countries prone to natural disasters, who would like a convenient and motivational way to create and easily manage an emergency stockpile of supplies.

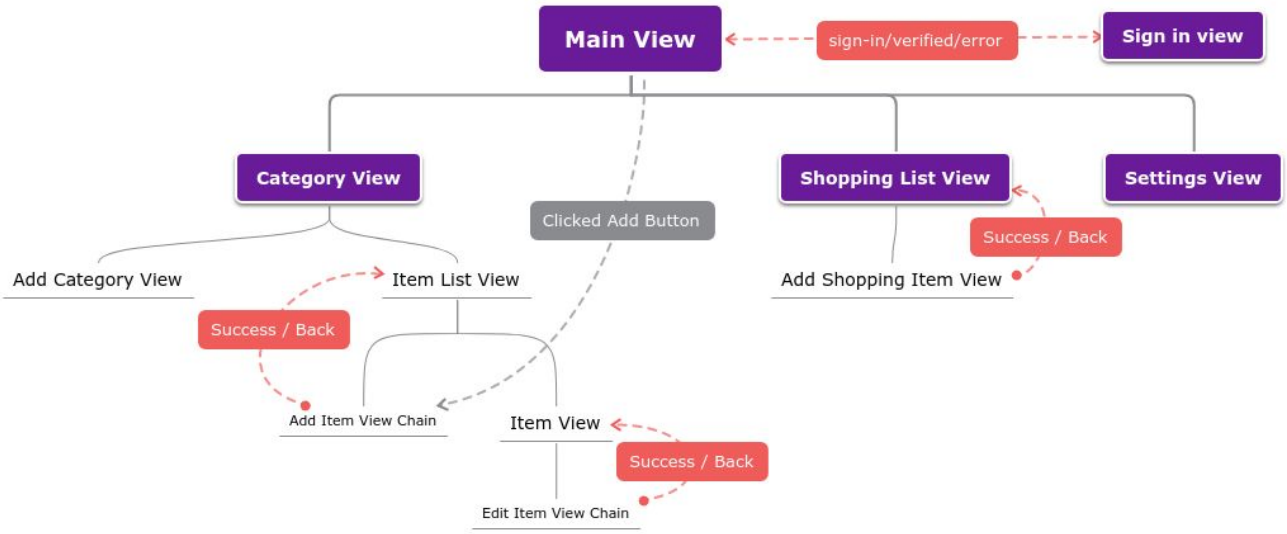
It can be an intimidating task when faced with the notion of preparing emergency supplies. This is especially true for people who have moved from a country which is not especially prone to natural disasters.

Features

- Users can create a database of stockpiled food-items
- Track perishable items, receiving alerts to replace the stock
- Track total stocked calories and water volume.
- Lists are synced onto the cloud allowing for back-ups

- Create shopping lists and receive monthly prompts to buy items.
- Take pictures of stockpiled items, catalogued for easy retrieval/replacement.

Basic User Interface Flow

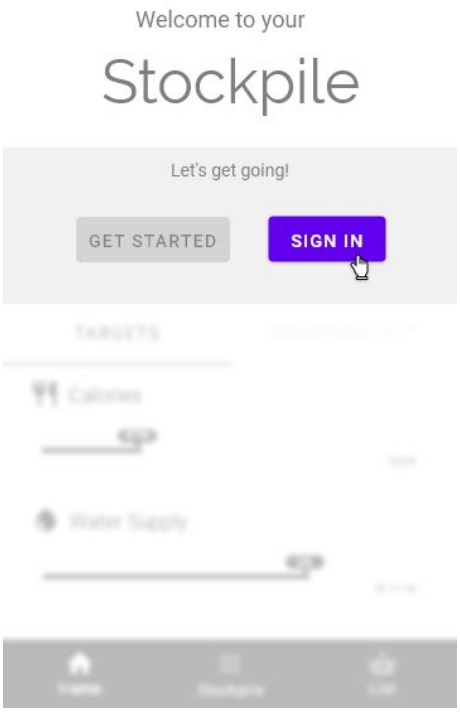


User Interface Mocks

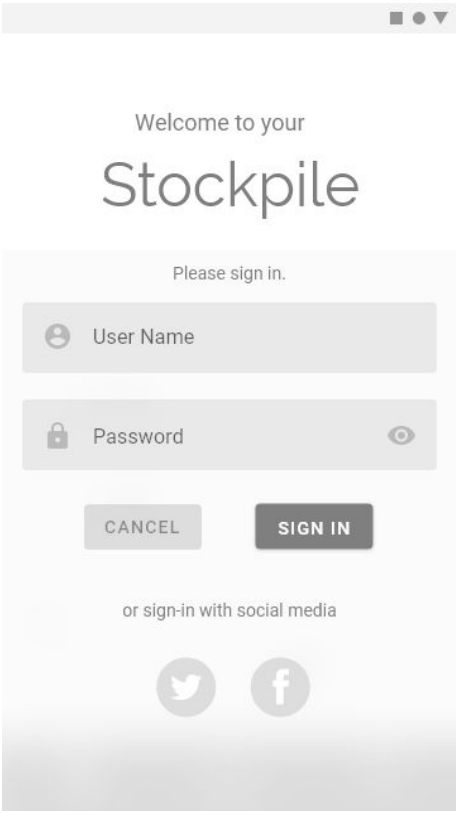
Screen 1 : Welcome Screen

The welcome screen allows user to begin using the application straight away or choose the sign-in option. A “teaser” of the app is shown to help indicate to users that no signin is necessary at this time.

Should a new user select sign-in they will be able to register straight away.



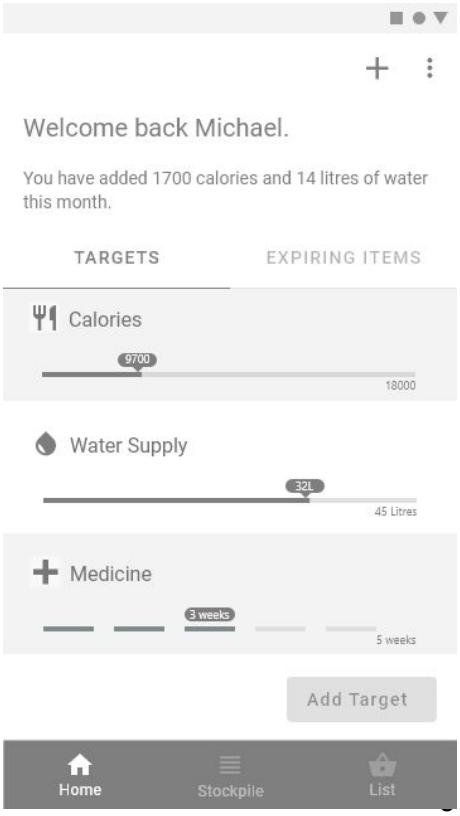
Screen 2.1 : Sign-in



The sign-in screen provides multiple options for user sign-in and registration. It should be simple and easy to access the application.

Note: While user data will be anonymously uploaded to the cloud, unless a user registers, migration or recovery of data would be impossible. The user should be made aware of this.

Screen 2.2 : Main / Home Screen



Personalisation - To add additional delight, a custom welcome message will greet the user and provide motivational information about how close they are to reaching their targets.

Targets - A visual representation of current targets set by the user. If a user has returned to the home screen from adding items, the home screen should show the progress that has been made in some fashion.

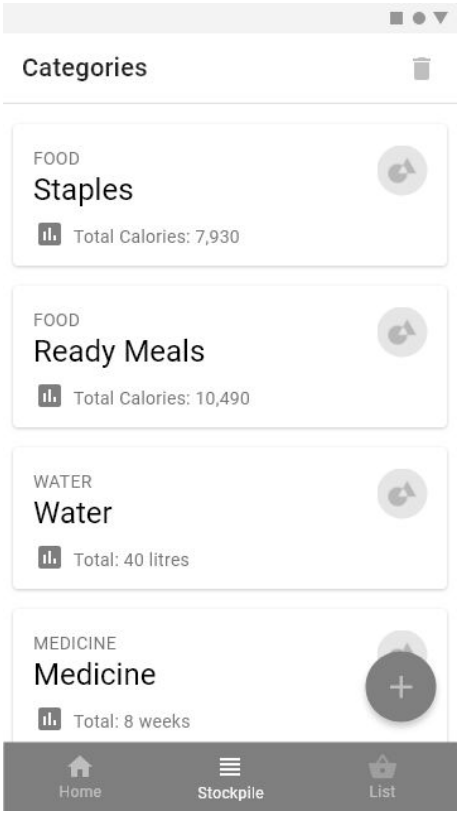
Expiring Items - An additional “quick look” tab allows the user to see all items which will be expiring in the near future. (When and how users want to be notified can be configured in the settings screen.)

Screen 3: Categories / Stockpile

Screen allows the user to browse through a list of categories. A set of default categories are provided but they are customisable.

Makes use of Material Design “Cards”.

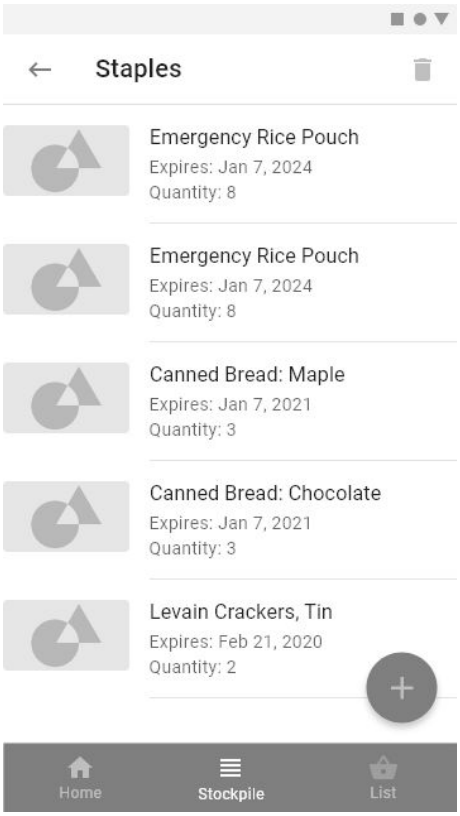
Floating Action Button - Introduces a dialogue box, allowing the user to create a custom category.



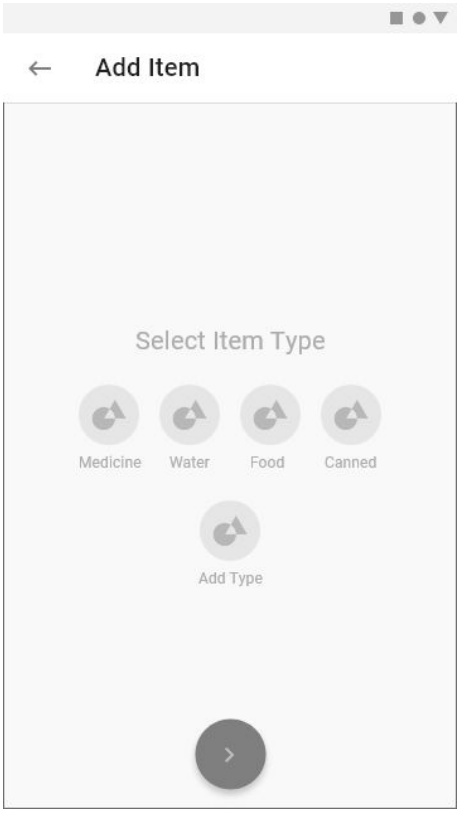
Screen 4: Item List View

Lists all items within a given category in a “list view”. Items can be managed from this screen (edited or deleted).

Floating Action Button - Takes user to a new screen allowing them to begin the process of adding an item. (See screen 5 - 7)



Screen 5.1: Add Item 1

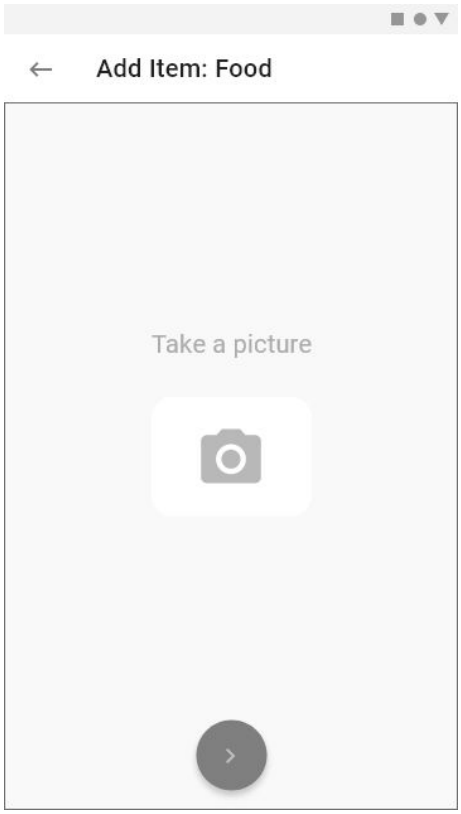


Note: This screen is only shown if the user has selected to add an item from the home screen. If the user adds an item from a category, that category will automatically be selected as the item type.

Allows a user to select which category the new item should belong to.

Floating Action Button: Takes the user to the next screen in the add item process.

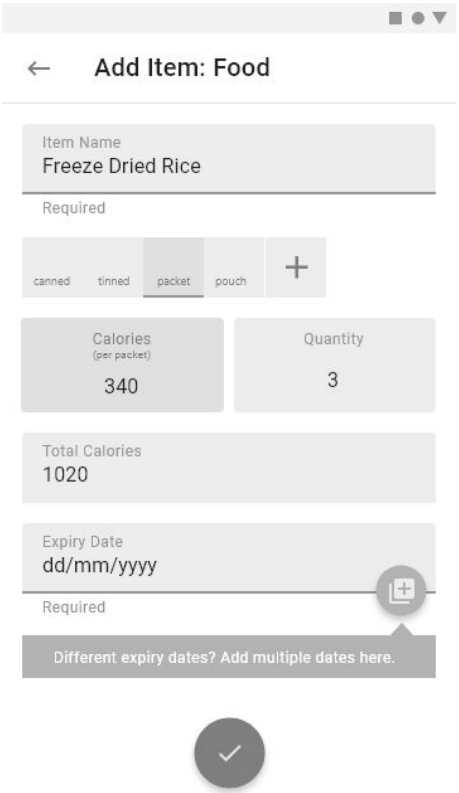
Screen 5.2: Add Item 2



This screen allows the user to take a picture of their item.

Floating Action Button: Takes the user to the next screen in the add item process.

Screen 5.3: Add Item 3

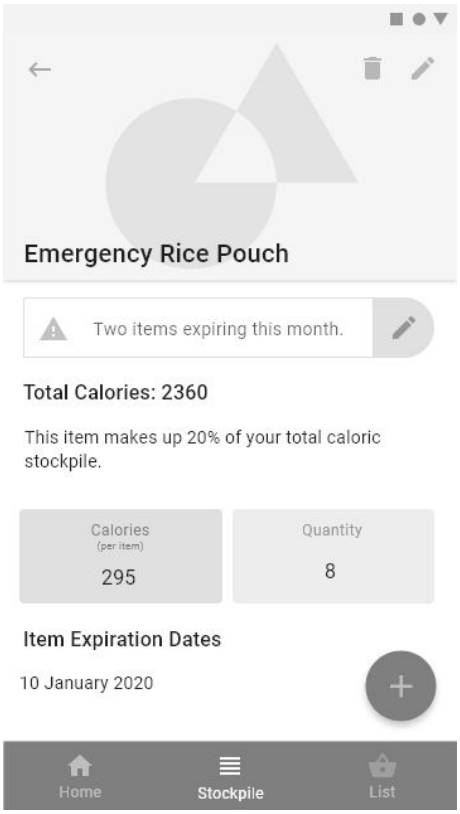


This screen allows the user to add the rest of the details for the item they are adding.

These details are specific to each category type, and adjust accordingly.

Floating Action Button: Completes the add item process taking the user back to the page they came from.

Screen 6.1: Item View

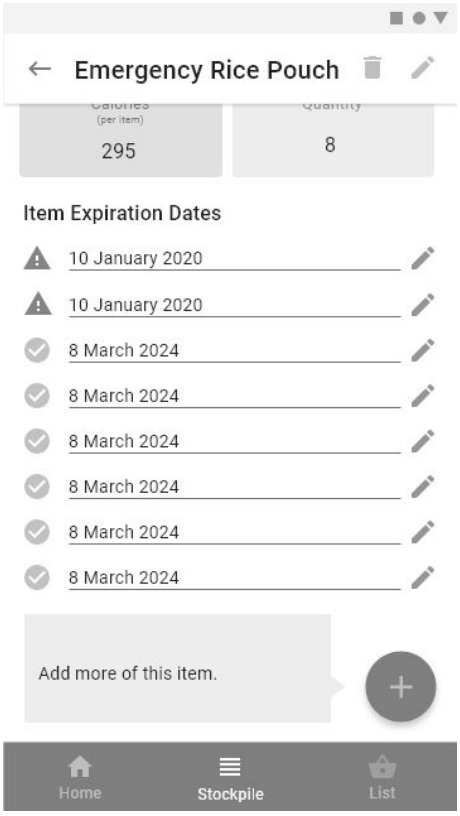


The user can view the details of a specific item from this screen.

Floating Action Button: Easily add items of the same type.

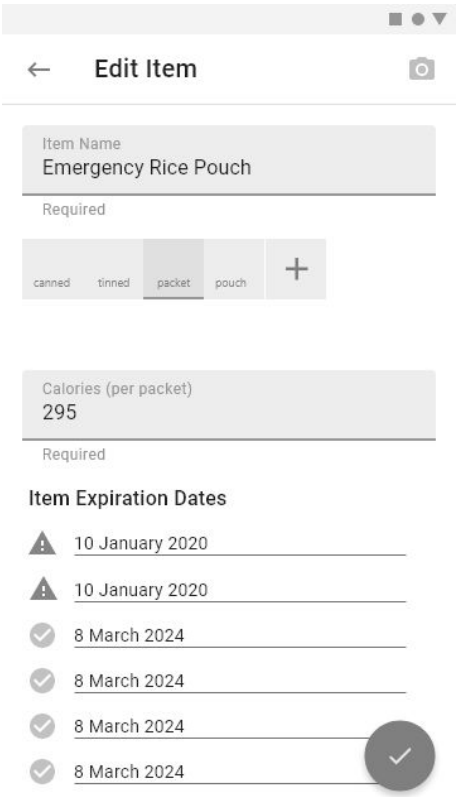
Screen 6.1.2: Item View (Extended)

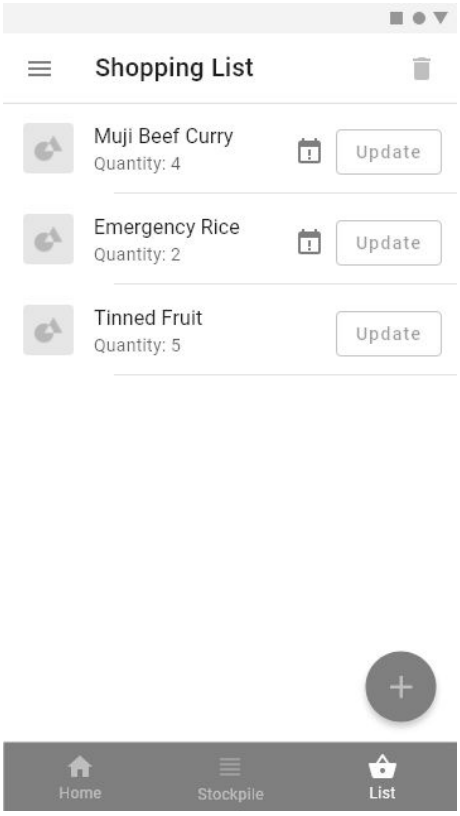
A user can directly edit expiry dates from this screen - should they be updating an item. Items can also be deleted from here.



Screen 6.2: Edit individual item

A user can edit the details of an item entry from this screen.

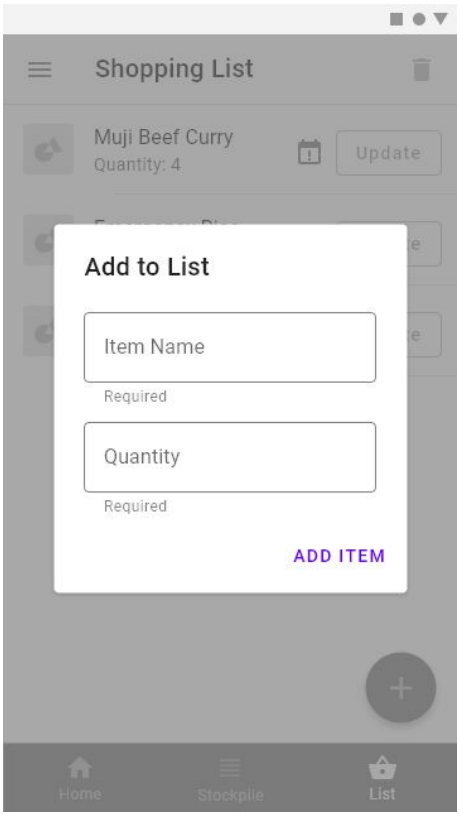




Screen 7: Shopping List

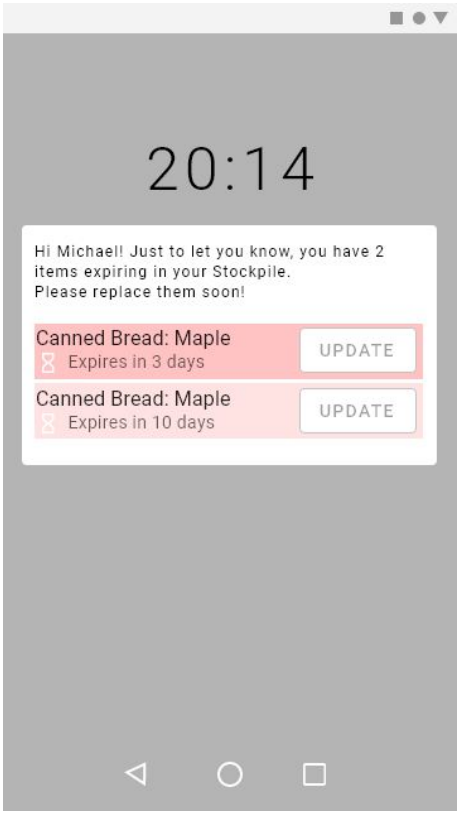
A convenient list which allows the users to see items which will soon expire. As will as then, custom items can be added to the list, which can then be updated and added to a category at a later date, if the user purchases that item.

Floating Action Button: Introduces a dialogue box allowing the user to quickly add a shopping list item..



Screen 7.1: Add Shopping Item Dialogue

At this stage in the item adding process, it was decided not to force the user to add superfluous detail to the shopping item, as the user should only have to input details if and when they buy a particular item.



Widget: Expiring Items

This is a collection widget, which shows soon to expire items. Colors will vary (tastefully) to indicate items which are closer to expiration.

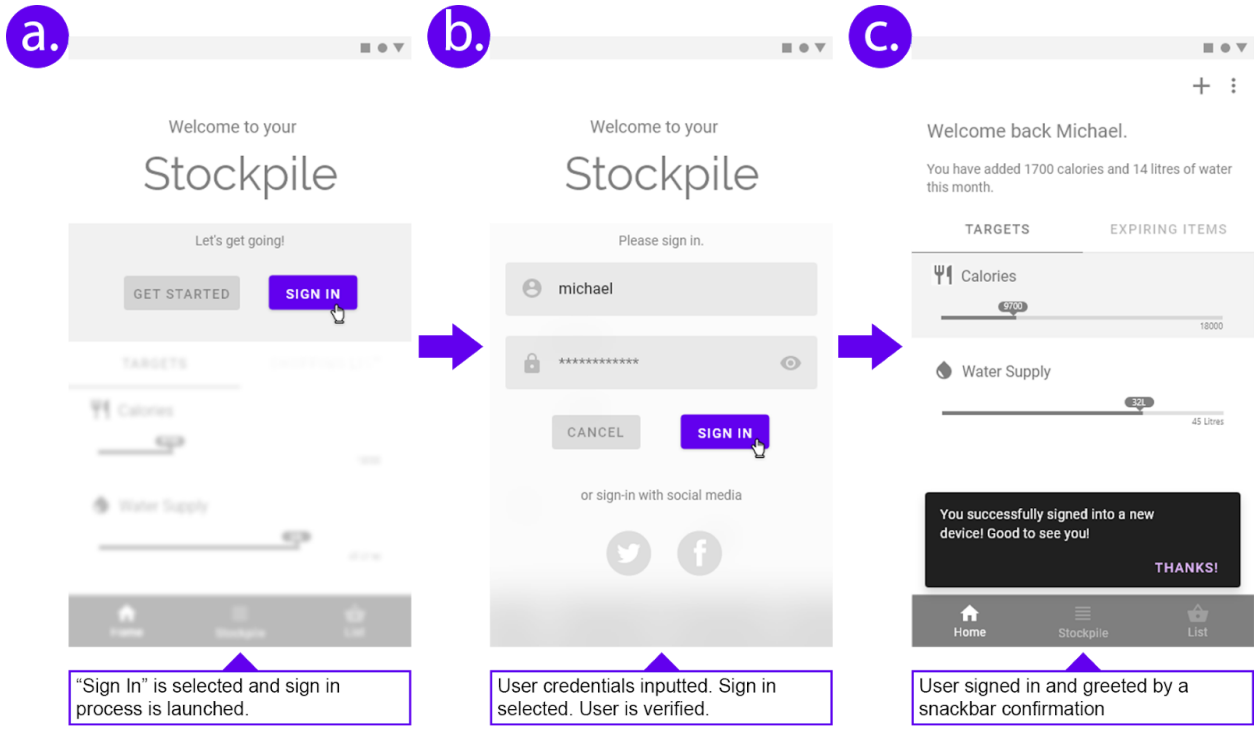
The user can use the update link to send them directly to that items “Edit Item” screen and make the required changes.

The list of items will be scrollable.

User Journeys

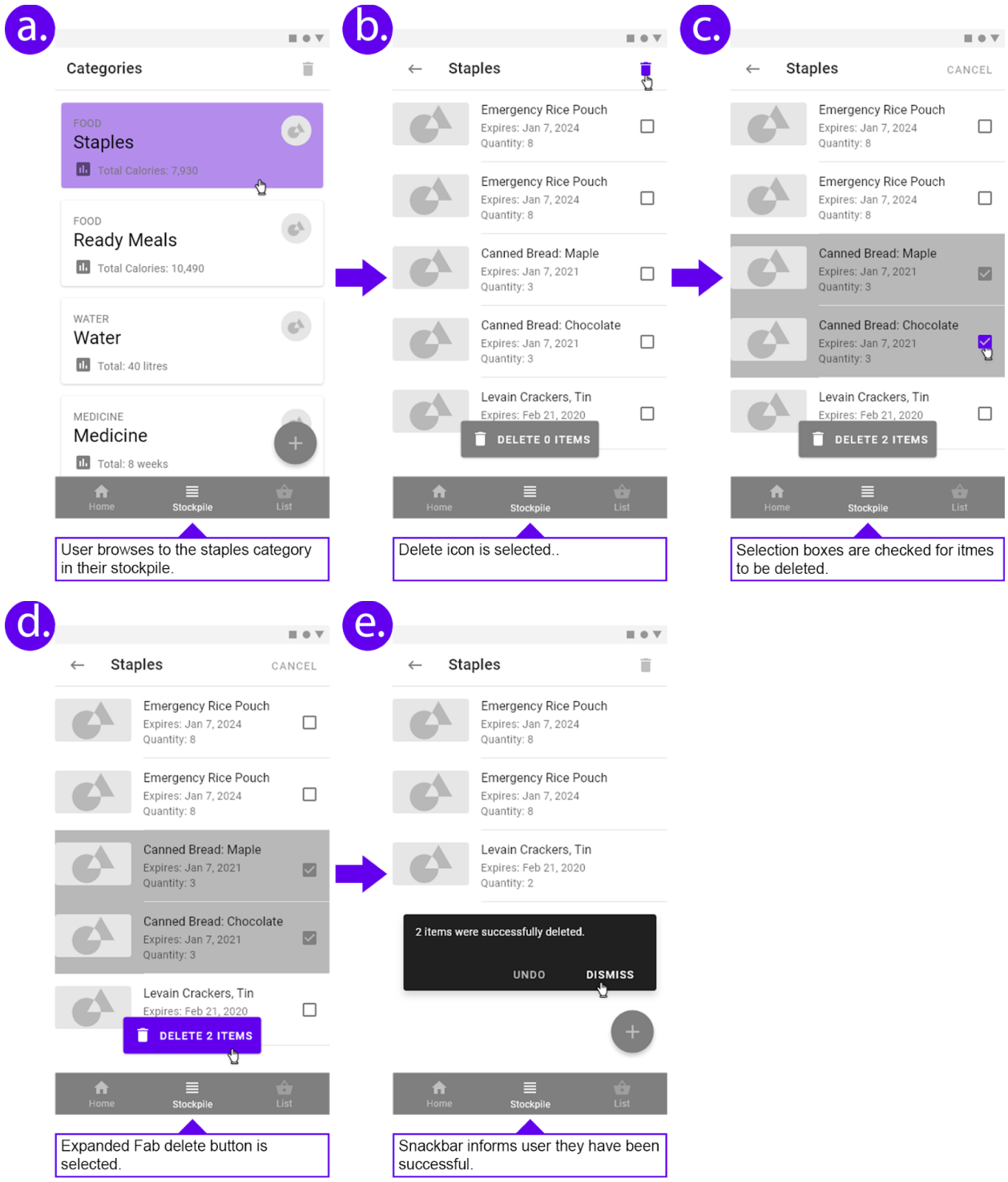
Signing in with a pre-existing account

A user who has been using “Stockpile” for some time has purchased a new mobile. They have registered an account using their google I.D. They will log-in to Stockpile and find their data is automatically migrated.



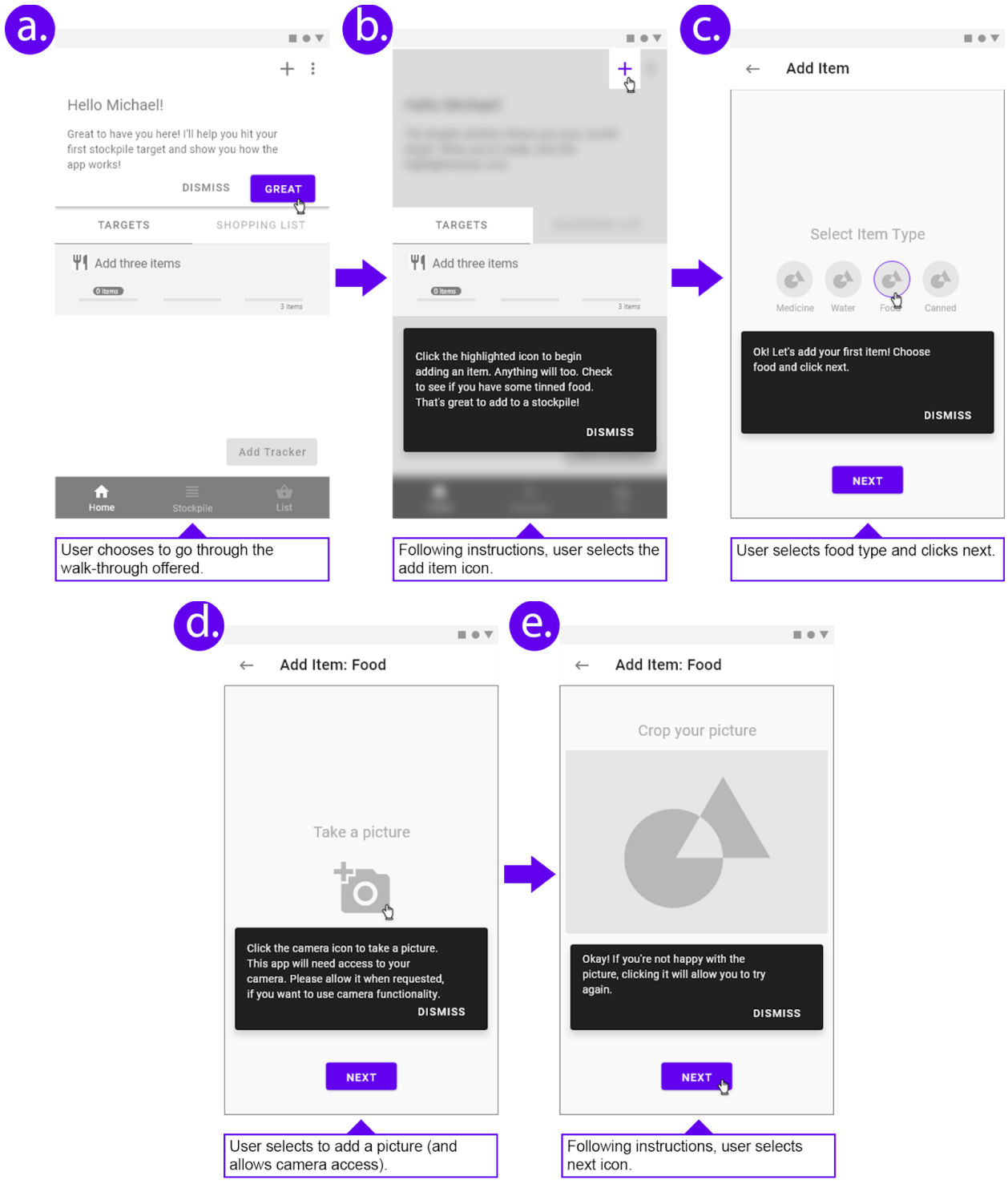
User deletes items from their stockpile

A user wishes to delete items from their stockpile inventory.

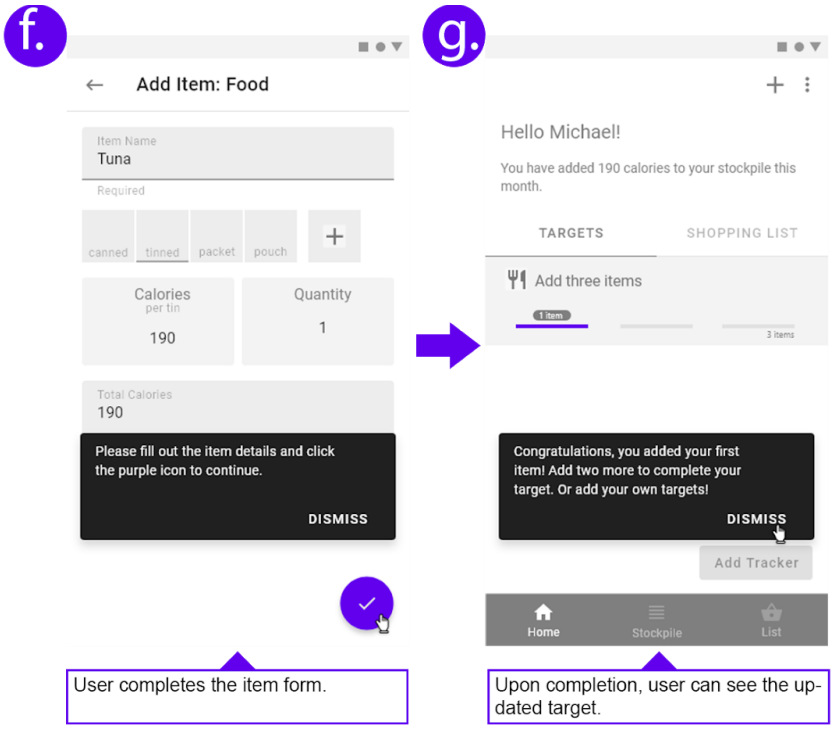


User signs in for the first time

When a user first signs in, a target is automatically added. That is, a target created for the purpose of showing the user how to add an item.



User signs in for the first time (continued)



Key Considerations

Programming Language

Stockpile will be developed using the Java programming language.

How will your app handle data persistence?

Data persistence will primarily be handled via the use of a Firebase Cloud firestore and enabling offline access (a very important feature of this app).

Local caching will be used to store changes a user makes and will be pushed to the realtime database when the application is removed from the recent app stack, or killed by Android.

See:
https://pamartinezandres.com/lessons-learnt-the-hard-way-using-firebase-realtime-database-c609b52b9afb

Images (of stockpiled products) will be persisted through a combination of Picasso library caching, local file storage and Firebase Cloud storage.

Anonymous authentication will be available, and if and when a user wishes to migrate their data, they will be able to do so by creating an account.

Describe any edge or corner cases in the UX.

Adding Items from the main screen

Screen 2 : Main / Home Screen
It will be possible to quickly add a new item from the home screen. Upon completing the addition of the item, the user will be returned to the home screen. At that point a toast will appear, informing the user of the successful addition of the item, and a link to add another item in that category.

Deleting all entries of a specific item

Screen 6.1.2: Item View (Extended)
Should a user clear all entries of a specific item (ie; that item would have a quantity of ‘0’), then the user will be given the option to delete the item entirely, or keep it so they may add new entries to it at a later date.

Handling Database Entries
It may become easy for a user to forget the items in their inventory. To prevent the insertion of multiple items, an auto-predictive entry form should be used in the 'Item Name' field of the 'Add Item View'. This will assist the user in preventing such errors.

***NOTE:** Using cloud firestore, this could potentially be an expensive operation (financially). Denormalising name data, and including a mapped list with the initial user auth query, may be a potential way to save access reads. Explore options, ensuring solutions are not detrimental to the end-user.*

Offline Access
When browsing offline, the user will be informed via a 'snackbar' that any edits or additions they make will not be synchronised until they are next online.

Migration
When using the application, the user will be made aware through a 'snackbar' that their data will not be recoverable or transferable across devices unless they log-in/create an account.

Describe any libraries you'll be using and share your reasoning for including them.

AndroidX** / **Jetpack

As opposed to using support libraries, this project will utilise AndroidX. This will allow the best backward compatibility available.

Several Jetpack components will be used including; architecture components (lifecycle, viewmodel, etc), AppCompat, design library components, etc

AndroidX Library	VERSION
androidx.lifecycle	2.1.0
androidx.appcompat	1.1.0
androidx.databinding	Bundled with gradle

Gradle for Android

AndroidX Library	VERSION
com.android.tools.build:gradle	3.5.0

Material Design (Previously part of support libraries, now part of AndroidX)

Library	VERSION
com.google.android.material	**1.0.0**

Picasso

This is an image downloading and caching library. It will be used for fetching, caching an1 displaying images. Since images do not take such a prominent place in this application, the smaller library of Picasso was chosen (as opposed to Glide, which has a much larger library size).

Library	VERSION
com.squareup.picasso	**2.71828**

Butterknife

A view binding library which automatically generates boilerplate code.This reduces code ‘bloat’ and improves readability and clarity.

Library	VERSION
com.jakewharton:butterknife	**10.2.0**
com.jakewharton:butterknife-compiler (annotationProcessor)	**10.2.0**

Dagger 2

A fast compile-time dependency injection framework. used for dependency injection - assisting in separation of concerns and therefore application testing.

Library	VERSION
com.google.dagger:dagger	**2.24**
com.google.dagger:dagger-compiler (annotationProcessor)	**2.24**

Espresso

Espresso will be utilised for UI testing.

AndroidX Library	VERSION
androidx.test:runner	**1.1.0**
androidx.test.espresso:espresso-core	**3**

Firebase

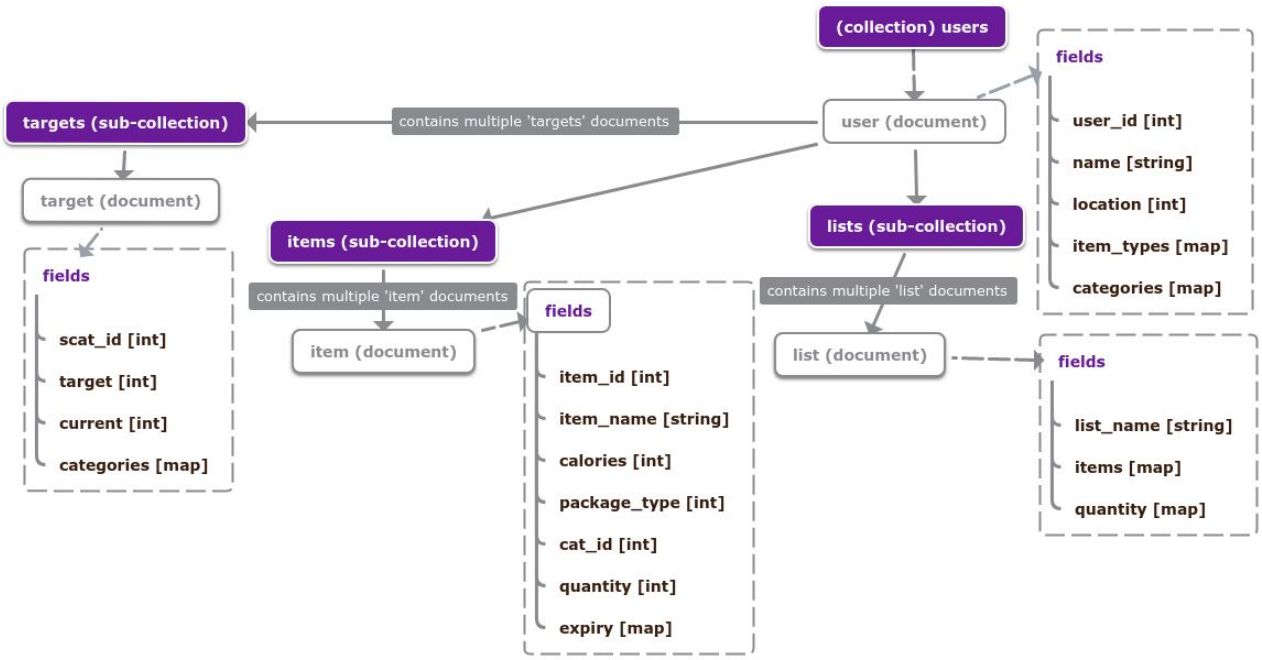
A. Cloud Firestore will be used as our database solution, to provide offline access, and to ensure future expandability and the ability for users to easily migrate across devices.
B. Firebase Auth will be used as our user authentication solution, to provide quick set-up and ease of access for users.
C. Firebase Cloud as the file storage solution

Firebase Library	VERSION
com.google.gms:google-services	**4.3.2**
'com.google.firebase:firebase-firestore	**21.1.1**
com.google.firebase:firebase-auth	**19.0.0**
com.google.firebase:firebase-storage	**19.0.1**

Describe how you will implement Google Play Services or other external services.

Cloud firestore

User data will be accessed and managed using the repository model (using MVVM). The following data structure is proposed (with expandability in mind):



[Note that repeated data-fields are to be expected using NoSQL database structures. Denormalization is used to improve query efficiency and reduce associated query costs, where no detrimental effect to the user is observed.]

Users could have a potential limit on the amount of items that they may add to their stockpile. This cap may be lifted, by an in-app purchase (or potentially a subscription model) allowing data costs to be covered.

Firebase Authentication

Authentication will be used in combination with other firebase services. Anonymous login will be enabled, to allow users to access the application straight away and register at a later date when migration / in-app purchasing becomes necessary.

Users will be authenticated through the Firebase API, and their unique user credentials (unique or otherwise) stored in the database.

Firebase Cloud
Users are able to store pictures of the items they add to their stockpile. These will be uploaded to firebase cloud storage, using an InputStream and Asynchronous upload.

See: https://firebase.google.com/docs/storage/android/upload-files

Next Steps: Required Tasks

Task 1: Project Setup

1. Initial project set-up
2. Configure libraries
3. Set-up simple ViewModel
4. Set-up simple test activity
5. Initial Dagger set-up
    a. Configure AppComponent, Modules and DaggerAppComponent
    b. Configure ViewModel injection, via a factory class (ViewModelFactory)
    c. Configure library component injection
6. Create a simple test module to test for successful view model injection.

Task 2: Implement Data Repository

1. Set up data model pojo's
2. Create mock-database
    a. Create fake database data in JSON
    b. Use GSON for parsing
3. Build the repository (using repository model)
    a. Create unit tests for repository
    b. Create repository (using repository-model), which will provide fake database data to ViewModel
4. Handle error cases

Task 3: Build Flavours

1. Create main build flavor and test build flavor which will use mock classes.

Task 3: Implement UI

[exclude input form functionality testing until firebase is implemented]
(Espresso tests will be created for all primary UI elements while developing UI.)

1. **Main / Home implementation**
    a. Main / Home Screen activity and fragment/s
    b. Implement features/functionality using the mock database.
    c. Create unit tests as required for testing lower level functionality
2. **Stockpile implementation (excluding all input form functionality)**
    a. Create the Stockpile activity and it's following children;
        i. Categories fragment
        ii. Item List fragment
        iii. Item View fragment
    b. During fragment creation, create espresso and unit tests as required.
    c. Input forms:
        i. Add Item activity (as a parent to the various add item views)
        Note: camera functionality to be tested by using local storage
        ii. Edit item activity

3. **Shopping List implementation**
    a. Create the List activity and fragments
        i. Shopping List fragment
        ii. Implement and test features/functionality using mock database
        iii. Create add item dialogue
    b. Create Settings activity
4. **Set-up data-binding,** using fake data (from repository) for testing
5. Ensure accessibility standards are met.
6. Ensure RTL switching is possible.

Task 3: Implement Firebase Auth

1. Set-up firebase authentication and allow for anonymous log-in.

Task 4: Implement Firestore

1. Create the necessary Dagger modules and components for Firestore
2. Set-up firestore filters, based on our data model
3. Create class to generate dummy-data to input to firestore
4. Modify repository to provide firestore data to ViewModel
5. Create unit and espresso tests to make sure functionality is as expected
6. Handle error cases appropriately.
    NOTE: Each user will have their own item database stored within their own "collection".

Task 5: Implement Firebase cloud storage

1. Implement necessary modules to integrate cloud storage with user auth and firestore,

Task 6: Implement input forms

1. Create test cases
2. Set up input form and dialogue functionality, inputting data to firestore
3. Implement special-case for adding a picture to an item
    a. Ensure when item pictures are taken, they are successfully uploaded to cloud storage, while locally caching a copy.
    This process will make use of an AsyncTask to resize the image to an acceptable resolution (to save on cloud storage and processing time when loading images). Use of an AsyncTask will all the user to continue the rest of their journey, inputting their new item.
    b. Handle item deletion - ensure files are removed from cloud storage
4. Handle error cases

Task 7: Create tutorial/walk-through

1. Implement user walk-through (refer to user journey: User signs in for the first time)
2. Create tests to ensure initial walk-through functions as expected

Task 8: Implement Widget

1. Implement simple widget, which allows users to see recently expiring items

Task 9: Final testing and configuration

1. Complete final testing and ensure functionality
2. Equip signing configuration and configure for Gradle installRelease build

[End of Proposal Document]