Udacity Android Nanodegree
Capstone Stage 1

Michael Vescovo

Contents

GitHub Username: mvescovo

Item Reaper

Description

Have things in your house you no longer need or use? Want to be more efficient with your shopping? Item Reaper is designed just for this purpose. It's job is to let you know when an item's time is up. It's the Grim Reaper for items. Know what you need and when. Prioritise based on your actual needs using your own data.

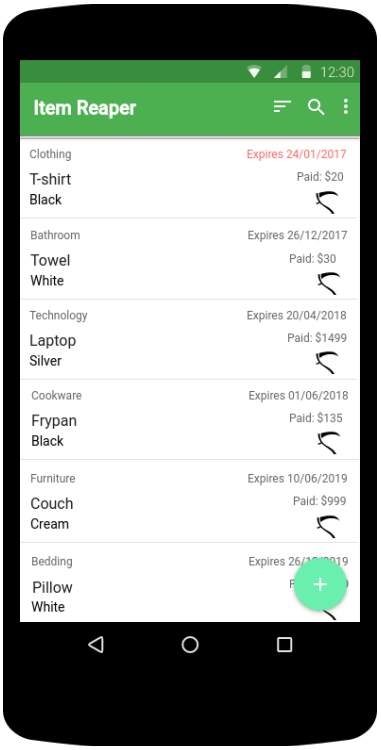
Intended User

This app is for anyone who might find it helpful to catalogue their own items. If you went shopping right now, what items would you need the most? How much did you pay last time? These are the types of questions Item Reaper intends to answer.

Features

- Add/edit items
    - Save item data to Firebase real time database
- Take a picture of an item or select an item with the picker
    - Save and retrieve picture using Firebase storage
- View list of items
    - Sort by expiry or purchase date
- Search for an item using any of the item details
- View item details
- Expire an item (hides from list), or delete an item
    - Undo button on snackbar
- About page including attributions for Reaper artwork (icons) and sound effect
- Includes widget to view the list of items
    - Can click an item to view details
    - Can click to create a new item
- Displays Admob ad in detail view
- Uses Firebase Analytics to try to understand user behavior within the app

User Interface Mocks



Screen 1

List of items sorted by expiry - the item expiring the soonest is displayed first. If there are multiple items expiring the same day the default behaviour will apply, since Firebase only allows one method of sorting.

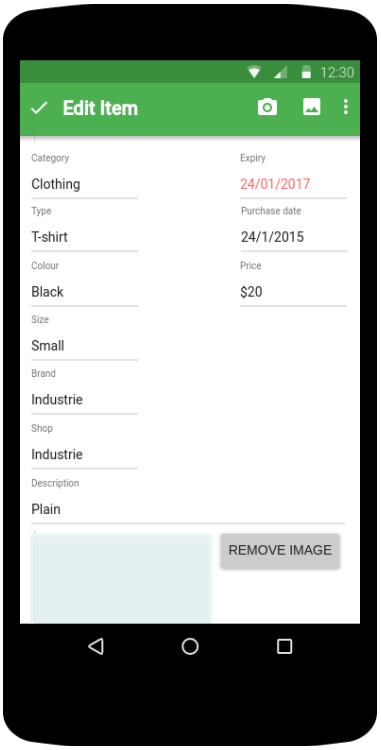
Each item includes the expire button to expire the item.

Items are displayed in tiles rather than cards due to homogenous content.

There is a sort option in the menu allowing for sort by purchase date.

There is a search option in the menu.

There is a FAB which links to the add/edit item screen.

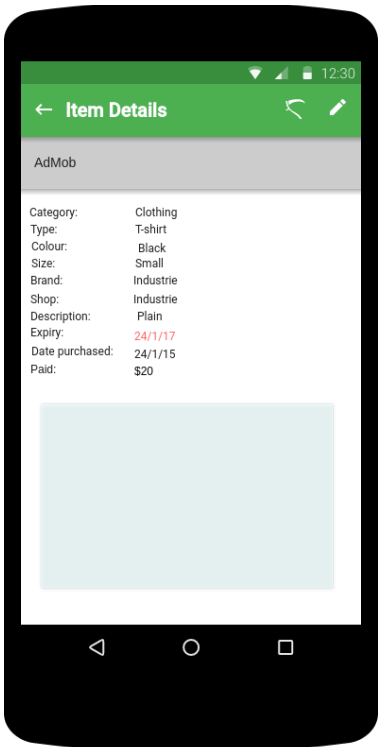


Screen 2

Add/edit an item for entering and saving item details. Arrangement of actual details screen will use a scrollview with one column. Mock shows two columns to reduce mock images. The order may also differ.

The user can click the camera menu option to take a picture using the camera, or click the select image icon to select an image with the picker.

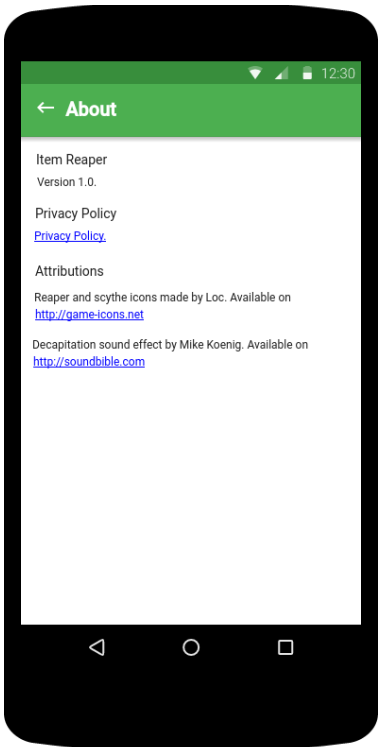
There is a remove image button.



Screen 3

View all item details including photo if available (photo is displayed in the mock as a blue rectangle). Actual details screen may differ in terms of order and layout of text but it will be similar.

Admob ad is displayed.

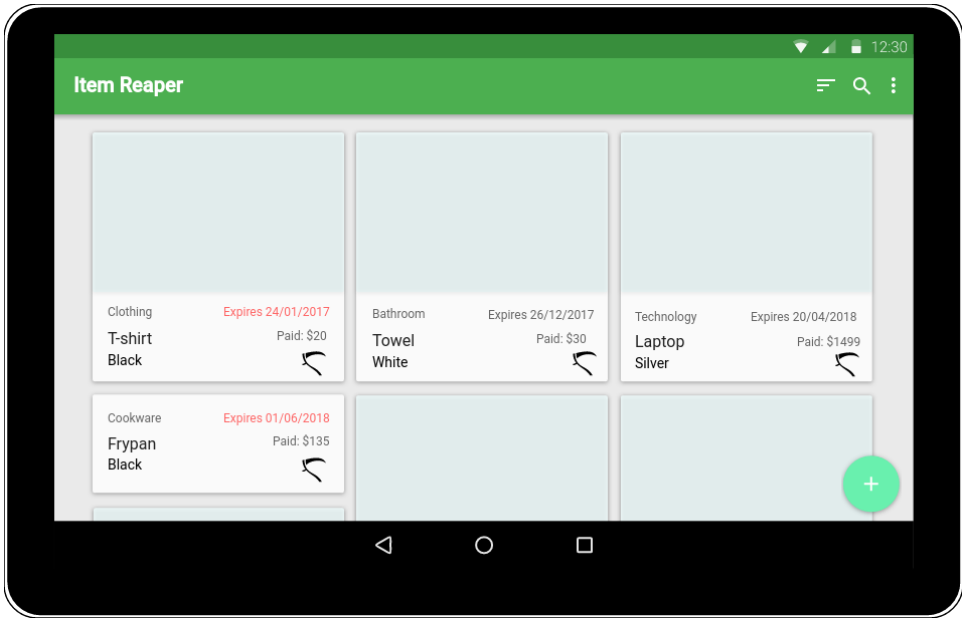


Screen 4

About page. Displays info about the app, link to privacy policy, and attributions for Reaper artwork (icons) and sound effect.

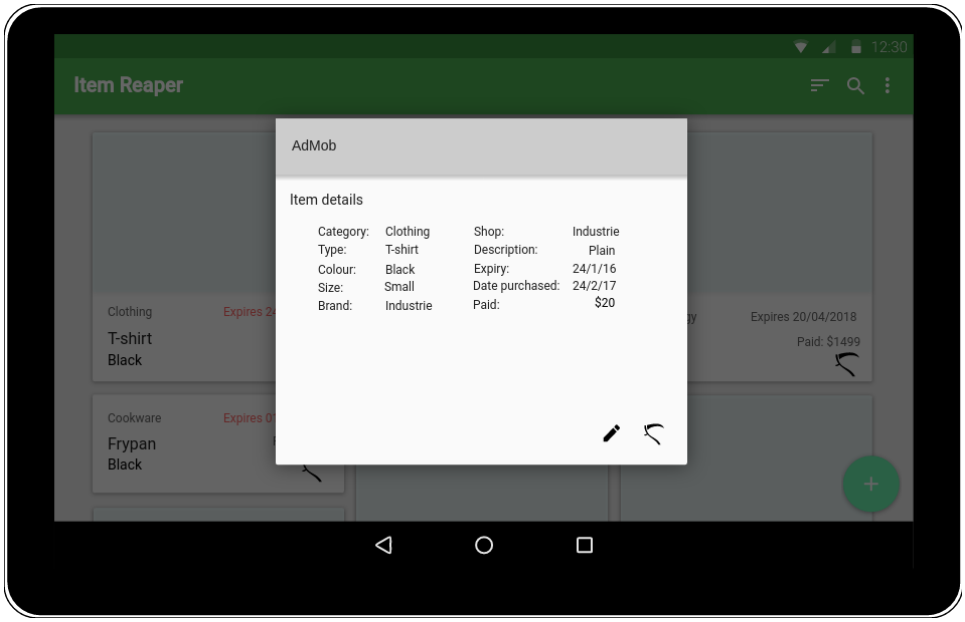
Screen 5

Item list on tablet using cards instead of tiles. Cards include the item image if available. This creates a more immersive experience on tablet sized screens.



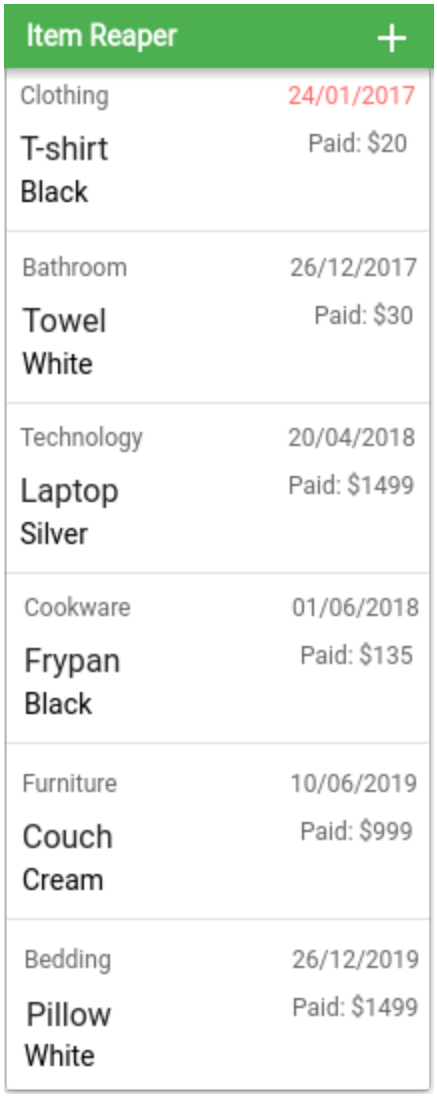
Screen 6

Item details and other screens on tablets. Items are shown in a dialog rather than a full screen Activity. This limits empty space as there isn't enough information to completely fill the screen. Actual layout of dialog may differ.



Screen 7

Widget for displaying the items in a list. Clicking on an item opens the relevant item details screen. Clicking the add icon shows the add item screen.



Key Considerations

How will your app handle data persistence?

The app will use Firebase real time database so the data is available across devices. Photos will be stored and synced using Firebase storage.

Describe any corner cases in the UX.

When editing an item in the tablet version this will be done in a dialog. I need to make sure the dialog is not able to be dismissed by clicking the surrounding area. The user must explicitly press the done button. Otherwise the user might lose their place while editing an item.

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

- Google support libraries for using AppCompat, annotations, design library, recyclerview, and cardview.
- Google play services as mentioned below for Google account login, Admob, and Firebase Analytics.
- Firebase for using the realtime database and storage.
- Junit for unit testing.
- Espresso and Mockito for UI testing.
- Butterknife for binding views.
- Dagger2 to get more control over class creation. In case I need to modify class constructors, or use different versions of classes under different situations such as when testing, it can make the process much easier. It probably won't be all that helpful in version 1 of my app, I'm adding this one here mainly because I want to learn it.

Describe how you will implement Google Play Services.

Google Account login. This is passed to Firebase auth to grant the user access to the real time database and storage.

Google AdMob. This is a great way to try to make money from the app should it be successful, and wouldn't it be great to be able to work on our own apps for a living. Ads can work in this app without disrupting the user experience. When a user views an item they will see one add above the item details.

Firebase Analytics. To learn how users are using the app. I could then make changes to future versions using this data, to try to make it even more useful.

Required Tasks

Standard feature subtasks

- Loop until all presenter tests are complete and passing:
    - Start writing a presenter test until it fails (including compilation).
    - Implement just enough that the test passes.
    - If the test is not complete:
        - continue writing the test until it fails again.
        - Implement just enough that the test passes again.
- Repeat the above procedure for view tests.
- Create tablet layout.

Task 1: Project setup

- Create GitHub repository.
- Create hello world Android Studio project.
- Set up Travis CI.
- Add app launcher icon.
- Create prod and mock flavours.

Task 2: Data

- Create data objects.
- Create repository tests in the same manner as standard feature presenter tests.

Task 3: About page

- Standard feature subtasks

Task 4: Add/edit items

- Standard feature subtasks

Task 5: View items

- Standard feature subtasks - include loader to meet rubric requirement
- Add search function that filters the list as the user types and uses AsyncTask - for rubric requirement

Task 6: View item details

- Standard feature subtasks

Task 7: Authentication

Up until this point all testing should have been done with fake data. Now it's time to connect to the Firebase real time database and test the prod flavour.

- Standard feature subtasks

Task 8: Widget

- Standard feature subtasks

Task 9: Display Admob ads

- Add Admob ad to view items list feature
- Add Admob ad to view item details feature

Task 10: Firebase Analytics

- Add Firebase Analytics to the app