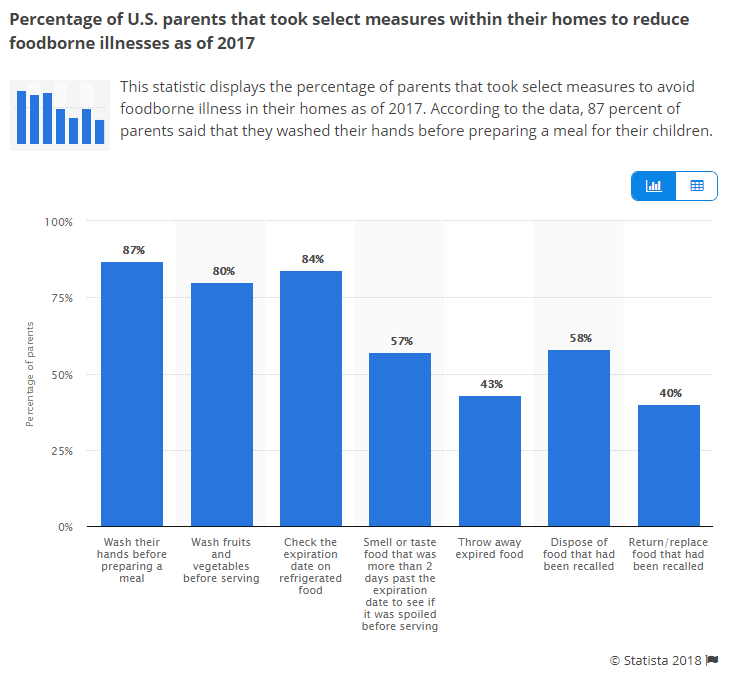
Expiry Tracker

Research



https://www.statista.com/statistics/877291/parent-food-safety-measures-us/

More than 4-in-5 consumers check dates on their food before consuming, with almost half throwing away expiring food

<https://www.cnbc.com/2015/04/22/americas-165-billion-food-waste-problem.html> (news link)

2. Don't put too much stock in the expiration date.

There is a great deal of confusion about expiration dates, and contrary to popular belief, most dates aren't statements about food safety.

There are two kinds of dates that commonly appear on food products. A "sell by" date is intended to be a message from the food manufacturer to the retailer so the store knows how long to display an item. It indicates that the product will still have significant shelf life once it reaches a consumer's home. "Best by" dates refer to quality, not safety, and signifies best flavor or peak freshness. A product will still be edible for several days afterward.

Read MoreA chef going where fast-casual boom has failed: healthy fare

Unfortunately, due to the lack of federal regulation about date labeling and confusion among consumers, many retailers and consumers throw food out on or before the date on the package, no matter what the date was intended to mean. "This contributes to enormous food losses at home and in the store," Berkenkamp said. "The best bet for consumers is to use your own judgment about whether the food in your fridge is still good. And if you think you can't use it up soon enough, pop it in your freezer rather than throwing it out," she said.

Github: <https://www.github.com/trod-123>

Description

Tired of throwing away food because you discovered they’re expired? Did you know that some foods are perfectly okay to use even beyond their printed date? What if there were a way to track your food and get notifications on when they’re about to expire?

*Expiry Tracker* allows you to enter expiration dates for your food products and reminds you when your food is about to expire. This app is designed to be interactive, so can enter your food with your date assistant. *Expiry Tracker* also provides additional information about how long your food can last even beyond the food’s printed expiration date.

Audience

Food-conscious consumers who

1. Care about the quality of the food they eat
2. Want to be more aware of whether their food is still good to eat
3. Wish to save money from throwing away food that’s still safe to use

This app is also tailored to store owners who need to

1. Keep track of the “sell by” dates for their products
2. Be notified in advance when their products will expire
3. Wish to prevent expired products from getting into the hands of consumers

Features

* Saves product and date information locally and in the cloud. Access your food information offline and online across your phones and tablets *(currently, only Android)*
* Barcode scanner to simplify putting food into the database. Pulls your food info quickly so you don’t need to enter it yourself! Also fetches images of your food online so you don’t need to take pictures
* Interact with your date assistant who can help store your food info for you
* Voice recognition to speed up food input, instead of typing it in
* Notifications to remind you when your food is about to expire. Customize how often and when you want to be notified before your food expires
* Home screen widget to display a list of foods that are expiring soon

Key Considerations

How will your app handle data persistence?

This app uses Room database to locally cache user data. For online storage, it leverages Firebase Realtime Database, which users can access through Firebase Authentication.

Food and product data received through barcode scanning and item search is gathered from the [UPC Item Database](https://devs.upcitemdb.com/) as Json objects, which are then parsed into Food objects which are stored in users’ own databases.

Ancillary data on how long food is still good for beyond the expiry date is gathered from [Eat By Date](https://www.eatbydate.com/) and returned to the user in the form of search results on a webpage. These pages can be bookmarked by the user and linked with their items.

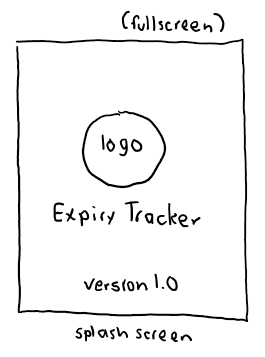
Describe any edge or corner cases in the UX

**User Interface Mocks and Behaviors**

User flow

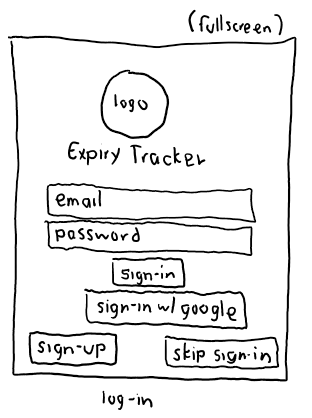
Notification / 
Widget 
Splash 
Log-in 
Sign-up 
Edit item 
At a glance 
Item detail 
List 
Review 
Add item 
captured 
Scanning 
items 
Global 
settings 

Splash screen



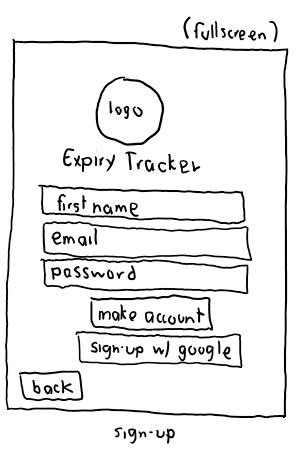
* An easy fade-in to the logo screen greets the user upon launching the app. The current version number is shown below the logo
* Shown whenever the app is launched from the Launcher
* Shown for around < 2 seconds before proceeding to next screen (similar to official Reddit app)
* Behind the scenes:
  + Prepare app resources (listeners, data, etc)
  + Check whether the user is signed in or not (either through Firebase or SharedPreferences):
    - If not signed in, go to log-in screen
    - Otherwise, skip to At a Glance activity

Log-in screen

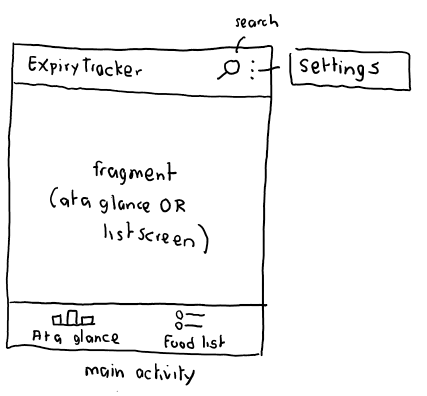


* Upon the device’s first launch, users can optionally log-in to their account by entering their e-mail and password, or by “logging in with Google”
* If coming from the Splash screen, the logo moves vertically seamlessly from the center position in the Splash screen to the upper position in the Log-in screen via SharedElementsTransitions
* Unregistered users can click on “sign-in” to create a new account
* Those without an account can also proceed without signing up
  + Internally, use SharedPreferences to indicate whether user chose not to make an account so user does not have to log-in again
* Always shown whenever the app is launched and the user is not currently logged in. If the user is already logged in, or decides to proceed without making an account, and the app is launched, this screen is bypassed.
* Libraries used:
  + Firebase Authentication
  + Google Sign-in
* *Note: Log-in is not required for saving on-device, but it is required for saving to the cloud*
* UI handlers:
  + E-mail EditText
  + Password EditText
  + Sign-in Button
  + Log-in with Google Button
  + “New Account? Sign-up!” Button
  + Skip sign-in Button

Sign-up screen

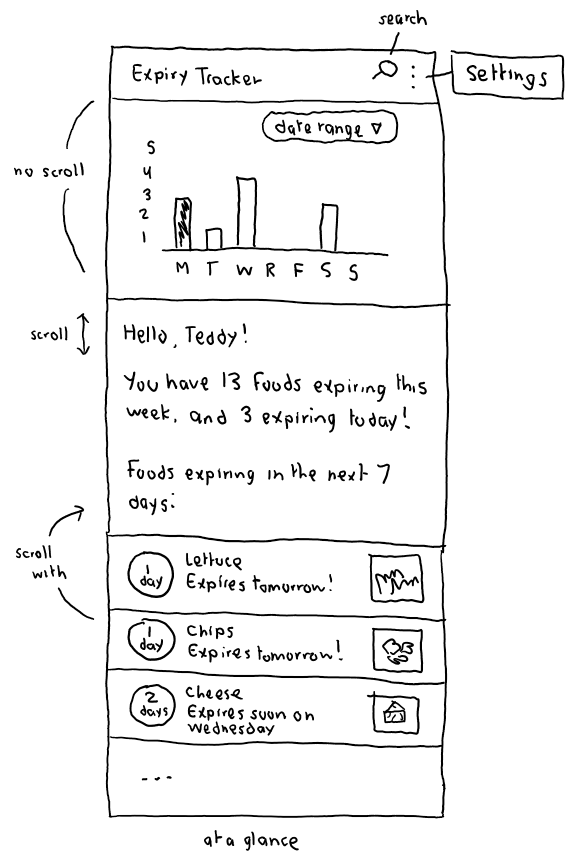


* Where users can create a new account using an e-mail address as credentials
* Looks very similar to the Log-in screen, but with an additional “Name” field and a “Confirm password” field
* Libraries used:
  + Firebase Authentication
  + Google Sign-in
  + Glide – for logo loading
* UI handlers:
  + Name EditText
  + E-mail EditText
  + Password EditText
  + Sign-up Button
  + Sign-up with Google Button
  + “Go back” Button
    - Return to the Log-in screen

Main Screens

* The Main Activity comprises of:
  + A. At a Glance screen
  + B. Food list screen
* Make up a horizontal ViewPager with tabs shown at the bottom of the screen
* Each screen bleeds fully to the left and right edges, with no visible gap between screens
* Toolbar menu actions hold for all screens:
  + Settings (never show icon)
  + Search (show icon if room; icon = magnifying glass)
* For RTL layouts, screens are shown in reversed order (At a Glance is on the right instead of left)

A. At a Glance screen



* Greets the user with a summary for the current week, detailing how many foods are expiring soon, via a bar chart and a short descriptive message below it. Also provides a small list showing which food is expiring in the next few days
* Data filter specs:
  + By default, shows data for the next 7 days, including the current date
  + User can set the date filter through the “date range” menu (won’t save across sessions)
    - Next 7 days
    - Next 14 days
    - Next 30 days
  + Applies to both bar chart and list, to ensure that both are properly sync’d
* Bar Chart specs:
  + X-axis shows dates (Mon, Tue, Wed, etc)
  + Y-axis shows # of foods expiring that day (expressed as # of *items*, independent of actual *quantity*)
    - Axis ranges determined dynamically, based on day with the highest number of expiring foods
  + Each graph shows 1 weeks’ worth of info
  + Layout and appearance similar to Android’s Digital Wellbeing app
  + The bar corresponding to the current date is highlighted – this bar will always be the first bar by default
  + Tapping on each bar displays a bubble showing the date (Month Day) and the number of expiring foods for that date
  + User can indicate the date range to filter their data
* Descriptive TextView specs:
  + One sentence summary in the form:

[GREETING: e.g. Hello, Welcome back, Good morning], [USER NAME: shown only if user made an account]!

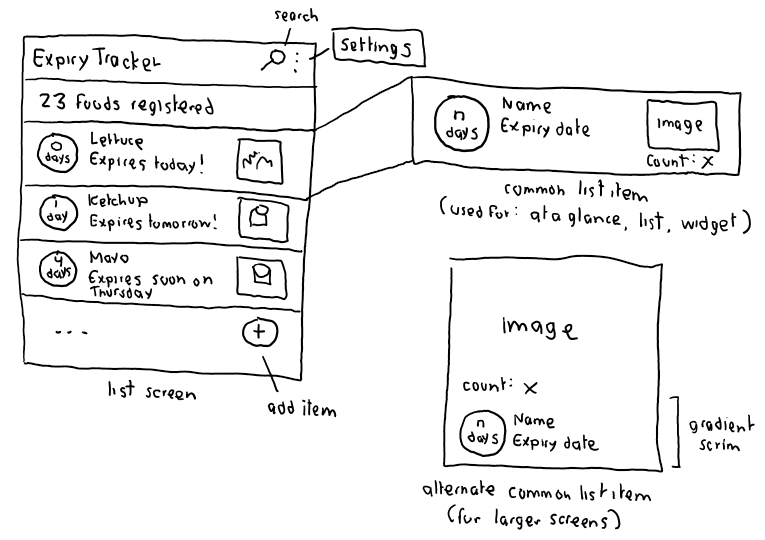
You have X foods expiring [DATE RANGE: e.g. this week, in the next 2 weeks, in the next 13 days], with Y foods expiring today! [X = the sum of the # of foods in the Bar chart]

Foods expiring [DATE RANGE]:

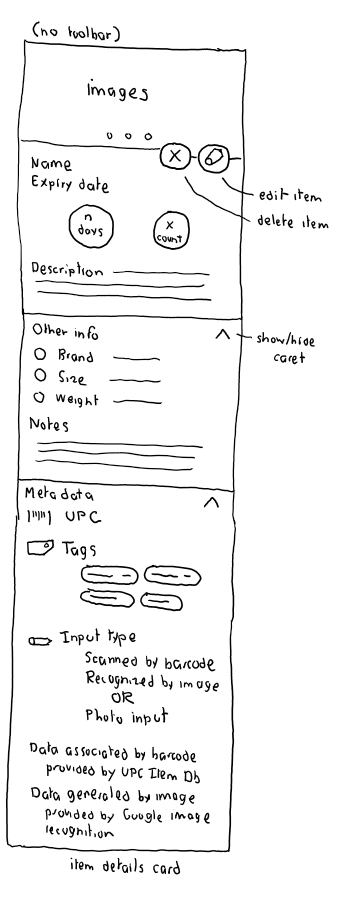
[SHOW LIST HERE]

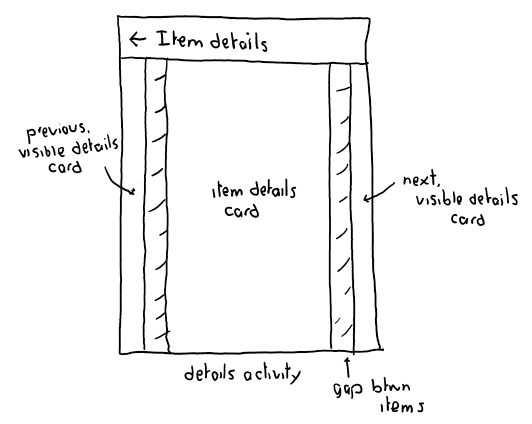
* + Sentence summary generates with a random greeting based on time of day, the user’s name, and the data filter set
  + If user had bypassed the log-in, then [USER NAME] is not shown
* List specs:
  + Horizontally-scrolling RecyclerView
  + Shows upcoming expiring foods for the next X days, specified by the data filter
  + Data is sync’d between the Bar Chart
* Scrolling:
  + Bar chart remains static on the screen
  + Descriptive TextView and the List scroll together behind the Bar chart
* Libraries used:
  + MPAndroidChart – for the bar chart
  + Lifecycle, Room, Paging, and RecyclerView – for the list of soon-to-expire foods
  + Glide – for image loading and caching

B. List screen



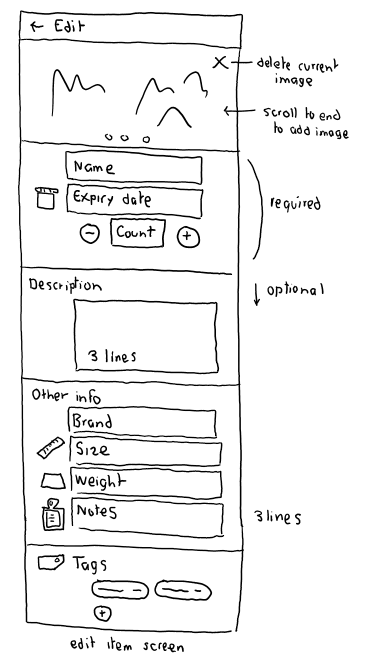
* Shows the list of all the foods users added to their database
* Above the list shows the number of foods added
* Each item contains the following views:
  + Food image, in a circular frame
  + Name
  + Expiry date, formatted based on # of days to that date
    - If it’s today, formatted as “Expires today!”, in red color font
    - If it’s tomorrow, formatted as “Expires tomorrow!”, in orange color font
    - If it’s within the next 7 days, formatted as “Expires soon on [DOW: e.g. Monday]”
    - If it’s within 8-14 days, formatted as “Expires on [DOW] next week”
    - If it’s 15 days but still within the current month, formatted as “Expires on the [ORDINAL DATE: e.g. 4th]”
    - If it’s 15 days but in the next month or later, formatted as “Expires on [MONTH DAY: e.g. Feb 25]”
  + Quantity, shown if more than one, in the form “Count: X”. Otherwise, hidden
  + # of days before expiry date, in a circular frame, with the color of the frame based on the number of days
    - If it’s today, color is red
    - If it’s tomorrow, color is orange
    - If it’s the 3rd day from today, color is yellow
    - Otherwise, color is default
* Clicking on a food item shows the Detail screen for that food item
* By default, list is sorted by earliest expiration first
  + Will only sort in this way – for this app, it doesn’t make sense to sort in any other way
* Swiping either left or right removes the item from the database
  + Upon swiping, Snackbar appears from bottom, allowing users to UNDO the operation (duration: Snackbar.LENGTH\_LONG)
* Floating action button lets users add new food items
  + Anchored to the bottom | end of the layout
  + Shows only when user is not scrolling, otherwise hides
  + Clicking on the button prompts 3 smaller floating action buttons to appear:
    - Camera icon button: add new item via camera
    - Image icon button: add new item via image already on device
    - Text icon button: add new item without image (go straight to a blank “edit” item screen, called “Add item” instead)
* “Continuous” shared elements transitions with Detail activity, only for the clicked list item
  + Image only
* Libraries used:
  + Lifecycle, Room, Paging, and RecyclerView – for the list of foods
  + Glide – for image loading and caching

Detail screen

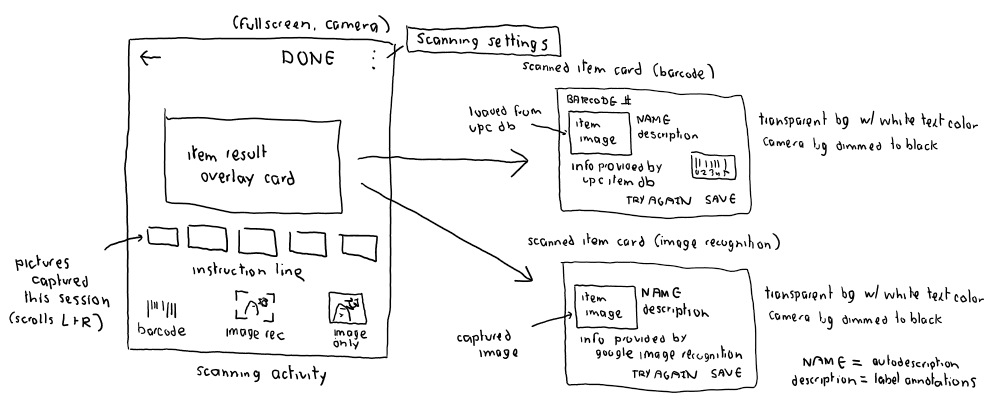


* Shows details about the selected food item
* Users can horizontally page through their item list through FragmentStateViewPager
* Each page contains enough horizontal padding to show a portion of the previous and next pages, to indicate to user that they can swipe left and right to scroll across pages
* Details contents:
  + Food images that the user can scroll through
    - Contains an indicator on the bottom of the images to show how many images there are, and the position of the current item
    - The end of the pager has an “Add” button that allows users to add more images by either
      * Picking an image in memory
      * Taking a picture
  + The first section contains main info:
    - Name
    - Expiry date
    - Number of days to expiration, in a circular frame
    - Quantity, in a circular frame
    - Item description (not shown if empty)
  + The next section is a collapsible section, titled “Other info” (not shown if all fields are empty):
    - Brand (not shown if empty)
    - Size (not shown if empty)
    - Weight (not shown if empty)
    - Notes (not shown if empty)
  + The last section is a collapsible section, titled “Meta data”:
    - Barcode number (not shown if empty)
    - Tags to aid in search and filtering
    - Input type – one of:
      * “Scanned by barcode”
      * “Recognized by image”
      * “Plain input”
    - Data attributions
* 2 floating action buttons:
  + Delete the current item
  + Edit the current item
* “Continuous” shared elements transitions when returning to List activity, only for the selected item. Upon return, activity will scroll to current location of item for a seamless transition
  + Image only
* Order of pages is reversed for RTL-enabled devices
* Libraries used:
  + Glide – for image loading and caching

Edit item screen / Review captured items screen

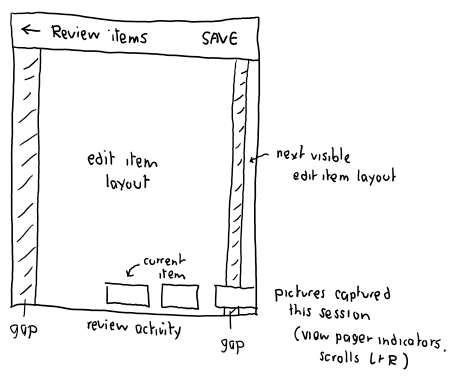


* Allows user to edit details of an item
* User can edit any field, except for the following fields provided in the Detail screen:
  + UPC/barcode number
  + Input type
* This screen is also used when reviewing items from the Add Item screen. Depending on the Capture mode, some fields may be auto-filled:
  + If coming from Barcode scanner, the following fields will be auto-filled if the information is available:
    - Name
    - Description
    - Brand
    - Size
    - Weight
  + If coming from Image recognition scanner, the following fields will be auto-filled if the information is available:
    - Name
    - Description
    - Brand
    - Tags
  + If coming from Image capture only, no fields will be auto-filled
* Libraries used:
  + Glide – for image loading and caching

Scanning screen (Add new item screen)

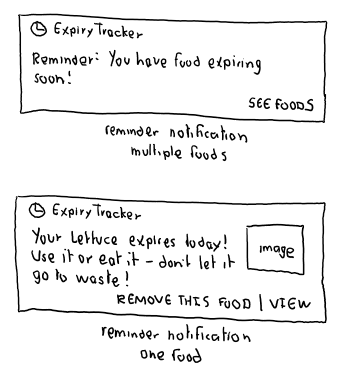
* Users are given the choice to add a new item via scanning a barcode, or taking a picture. This choice is provided via “Capture Modes” at the bottom of the screen
* The instruction line provides user instructions based on capture mode
  + Barcode: “Place the camera anywhere over a barcode to scan it”
  + Image rec and Image only: “Tap the screen to capture an item”
* Users can take as many pictures as they want in succession, using any capture mode. Saved pictures from a single session populate a horizontal list view above the capture mode buttons
* General scanning flow:
  + Picture taken
  + If using Barcode or Image Rec, data is fetched corresponding to the barcode or picture captured. In the meantime:
    - Empty transparent card overlay appears with loading spinner
    - Camera continues showing, but capturing is disabled
  + If using Barcode or Image Rec, transparent card overlay is filled with fetched item details
    - If there is no internet connection, then an error is shown, but users can still add the item to their database
    - No internet error: “We were unable to retrieve data for this item”
  + After the details are filled, depending on which details are already provided, the app prompts the user for the following, to which the user provides an oral/spoken response (by default, can be configured to text in settings):
    - Name
    - Description
    - Expiry date
  + Users decide whether to “Try again” to take another shot, or to “Save” the item to their current session
  + Captured image gets added to the horizontal list view, and card overlay fades out
  + Capturing is re-enabled
  + Take another picture and repeat, or select “DONE” to end the current session
* Order of items in horizontal list view are reversed for RTL-enabled devices
* Barcode overlay elements:
  + Barcode # (using Firebase ML Kit)
  + Name (from UPC Database)
  + Description (from UPC Database)
  + Item image (from UPC Database, loaded via Glide)
  + Data attribution string: “Info provided by UPC Item Database”
* Image Rec overlay elements:
  + Name (from Firebase ML Kit)
  + Description (from Firebase ML Kit)
  + Captured image (user taken, loaded via Glide)
  + Data attribution string: “Info provided by Google image recognition”
* Capture settings:
  + Auto-focus by default (SwitchPreference)
  + Flash on by default (SwitchPreference)
  + Beep when captured (SwitchPreference)
  + Vibrate when captured (SwitchPreference)
* Pressing the back button with pictures in the horizontal list view prompts the user with an alert dialog: “Are you sure you want to stop? If you need to save your items, make sure to press DONE. Otherwise, your items will be lost.”
* Pressing the DONE button takes the user to the Review screen
* Libraries used:
  + Firebase ML Kit – for barcode scanning, image recognition
  + Glide – for image loading and caching
  + UPC Item Database – for retrieving product info from barcodes

Review items screen



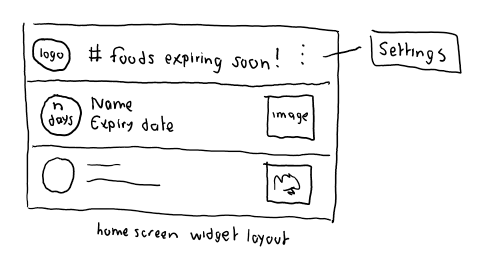
* This is where users review all the items they captured during their capture session.
* The Edit Item screen is reused here as an item in a FragmentStateViewPager. The horizontal list view on the bottom shows all the items captured, through which users can freely page through
  + The current item is shown in the center of the screen
* Each page contains enough horizontal padding to show a portion of the previous and next pages, to indicate to user that they can swipe left and right to scroll across pages
* On
* Pressing the back button takes the user back to the scanning activity, if the user wants to add more items
  + While the user edits each item, data will be saved in real time in a cache so user does not need to click “SAVE” for every screen
* Pressing the SAVE button sends the data into the database and takes the user back to the List screen
* Order of pages is reversed for RTL-enabled devices

Notifications



* Users are regularly notified daily in the AM as their food items’ expiry dates come closer
* There are 2 main notification layouts, based on how many foods are featured in the notification:
  + If multiple
    - A general, plural message is provided
    - A button “See foods” takes the user to a special List activity that shows only the foods the notification is referring to
      * This list activity looks exactly the same as the List screen, but its contents are filtered to just the referred food
  + If single
    - A custom message is provided, with the name of the food item that is expiring
    - Image of food
    - 2 buttons
      * “Remove this food” is a convenient button, allowing users to remove the food from the database through the notification
      * “View” allows the user to see that item’s detail screen
* Users can personalize how often they want these notifications to appear:
  + Receive reminder notifications? (SwitchPreference; default = off)
  + Be reminded when? (MultiSelectListPreference; disabled if receive = false; default = all checked):
    - Day of when foods expire
    - Day before foods expire
    - 2 days before foods expire
    - 3 days before foods expire
  + Time of day to be reminded? (ListPreference; disabled if receive = false; default = Morning):
    - Morning
    - Noon
    - Evening

Home-screen widget



* The app provides one home screen widget, which shows a list of the foods that are expiring in the next X days
  + List items look the same as list items in the List activity
  + List is sorted with the earliest expiry date first (unchangeable)
* Via the Settings action, users can set which foods appear in the widget, based on how many days are left before they expire
  + By default, widget will show foods expiring in the next 3 days
  + Users can choose any number of days between 0 and 7 days (ListPreference)

Global Settings (via PreferenceFragment)

* Notifications
  + Receive reminder notifications? (SwitchPreference; default = off)
  + Be reminded when? (MultiSelectListPreference; disabled if receive = false; default = all checked):
    - Day of when foods expire
    - Day before foods expire
    - 2 days before foods expire
    - 3 days before foods expire
  + Time of day to be reminded? (ListPreference; disabled if receive = false; default = Morning):
    - Morning
    - Noon
    - Evening
* Widget
  + Food to show based on # number of days until expiry date (ListPreference; default = 3 days)
    - Between 0 and 7 days
* Capture settings:
  + Auto-focus by default (SwitchPreference; default = on)
  + Flash on by default (SwitchPreference; default = off)
  + Beep when captured (SwitchPreference; default = on)
  + Vibrate when captured (SwitchPreference; default = on)
* Account settings:
  + Display name (EditTextPreference; default = blank, or the name the user provided when making an account)
  + Sign-in / Sign-out button (Preference)
    - Users who haven’t made an account and have data saved on device, but create an account at this stage
  + Wipe all ExpiryTracker data on device (Preference)
  + Delete ExpiryTracker account and all data on device (Preference; show only if currently logged in)

**Describe any libraries you’ll be using and share your reasoning for including them**

The implementation of this app will rely on several libraries. All of the libraries used will be the latest stable versions

Android libraries

* Android Support Library 27.1.1
* Android Architecture Components
  + [Lifecycle](https://developer.android.com/topic/libraries/architecture/adding-components#room) 1.1.1
    - For LiveData and ViewModel support with Room
    - android.arch.lifecycle:extensions:1.1.1
    - android.arch.lifecycle:compiler:1.1.1
  + [Room Database](https://developer.android.com/topic/libraries/architecture/adding-components#room) 1.1.1
    - For storing users’ data locally
    - android.arch.persistence.room:runtime:1.1.1
    - android.arch.persistence.room:compiler:1.1.1
  + [Paging](https://developer.android.com/topic/libraries/architecture/adding-components#room) 1.0.0
    - To be used with Recycler View to display users’ data in a list format, loading and displaying items efficiently on demand
    - android.arch.paging:runtime:1.0.0

Google Play Services / Firebase libraries

* [Firebase Realtime Database](https://firebase.google.com/docs/database/) 16.0.1
  + For providing online storage for users’ food items across their own devices.
  + Complements Firebase Authentication so users can only access their own content, and not anyone else’s
  + com.google.firebase:firebase-database:16.0.1
* [Firebase Storage](https://firebase.google.com/docs/storage/) 16.0.1
  + For users to save their pictures of their food online in a way so only users can access their own pictures.
  + Complements Firebase Authentication so users can only access their own content and not anyone else’s
  + com.google.firebase:firebase-storage:16.0.1
* [Firebase Authentication](https://firebase.google.com/docs/auth/) 16.0.3
  + For users to sign-in to their accounts to save and access their data across devices
  + Used with Google Sign-in
  + com.google.firebase:firebase-auth:16.0.3
* [Firebase x Google AdMob](https://firebase.google.com/docs/admob/admob-firebase) 15.0.1
  + For providing ads to the app’s free version only
  + com.google.firebase:firebase-ads:15.0.1
* [Firebase ML Kit](https://firebase.google.com/docs/ml-kit/) 17.0.0
  + For allowing users to take pictures of their products, either barcode or label, and parsing that data to simplify search and input
  + com.google.firebase:firebase-ml-vision:17.0.0
* [Google Sign-in](https://firebase.google.com/docs/auth/android/google-signin) 15.0.1
  + Allows users to sign-up and log-in using their Google account
  + com.google.android.gms:play-services-auth:15.0.1

3rd party utility libraries

* [Retrofit](https://github.com/square/retrofit) 2.4.0
  + HTTP client that simplifies gathering and parsing API results from online into JSON. Will be used for the APIs listed below
  + com.squareup.retrofit2:retrofit:2.4.0
* [Moshi](https://github.com/square/moshi) 1.6.0
  + For simplifying parsing JSON into Java objects
  + com.squareup.moshi:moshi:1.6.0
* [Glide](https://github.com/bumptech/glide) 4.8.0
  + For simplifying image caching and loading. The app will display images of the food items that users enter in the app
  + com.github.bumptech.glide:glide:4.8.0
  + com.github.bumptech.glide:compiler:4.8.0
* [Timber](https://github.com/JakeWharton/timber) 4.7.1
  + For simplifying logging and debugging
  + com.jakewharton.timber:timber:4.7.1
* [ButterKnife](https://github.com/JakeWharton/butterknife) 8.8.1
  + For reducing boilerplate code for view binding
  + com.jakewharton:butterknife:8.8.1
  + com.jakewharton:butterknife-compiler:8.8.1
* [LeakCanary](https://github.com/square/leakcanary) 1.6.1
  + For detecting memory leaks across fragments and activities
  + debugImplementation 'com.squareup.leakcanary:leakcanary-android:1.6.1'
  + releaseImplementation 'com.squareup.leakcanary:leakcanary-android-no-op:1.6.1'
  + debugImplementation 'com.squareup.leakcanary:leakcanary-support-fragment:1.6.1'
* [MPAndroidChart](https://github.com/PhilJay/MPAndroidChart) v3.0.3
  + For generating bar charts to portray numbers of expiring foods
  + com.github.PhilJay:MPAndroidChart:v3.0.3
* [PageIndicatorView](https://github.com/romandanylyk/PageIndicatorView) 1.0.1
  + Light library to indicate ViewPager’s selected page with different animations
  + Used in the Detail view to complement the food image pager
  + com.romandanylyk:pageindicatorview:1.0.1
* RapidFloatingActionButton 2.0.0
  + For easily implementing a Fab with “sub fabs”
  + Used in List item view for adding new items in different ways
  + com.github.wangjiegulu:rfab:2.0.0

APIs

Additionally, this app will rely on the following APIs:

* UPC Item Database: For grabbing data associated with barcodes and food

**Required Tasks**

Implementation

This app will be built on Java using the latest stable versions of Android Studio and the Gradle Build platform. It will target P, SDK 28, and support older versions back to K, SDK 19, to target at least 90% of devices. Gradle will be used to manage all app dependencies. See below for a list of the versions of software that will be used.

Software versions

* Android Studio 3.1.4
* Android Gradle plugin 3.1.4
* Gradle 4.10
* Java 1.7
* Minimum Android SDK version 19
* Target Android SDK version 28

The implementation will not use hardcoded values, except those defined in the res/values folder. This will hold true for dimensional values, strings, integers, Booleans, and so forth

This app will be localized through supporting RTL layouts for RTL languages, specifying which strings need translations, and providing flexible layouts to accommodate nuances of different languages

Additionally, this app will support Android’s accessibility features, through content descriptions and navigation through D-pad

This app will utilize responsive design to provide flexible layouts to accommodate the available space on the different screen sizes and densities of most devices used, including phones and tablets

Errors from server and user input, data parsing, and UI loading will be handled gracefully to prevent the app from crashing for users

A home screen widget will provide a list that updates daily, showing the foods that will expire the soonest

The user interface will follow Material Design guidelines as outlined in the official documentation [here](https://material.io/design/), also including fluid animations, transitions, and shared elements to provide delightful and engaging user experiences.

Free and Paid versions of the app will be available, with the free version containing non-obstructive banner ads. The Paid version will contain no ads.

This app will build and deploy a Release build configuration, and the source code will provide the associated keystore and passwords with the signing configuration used. The keystore will be referred to by a relative path.

Task 1: Project Setup