Expiry Tracker

Android App

Capstone Project for Udacity's Android Developer Nanodegree Program

Stage 1 Proposal prepared by Teddy Rodriguez, a Grow with Google Scholarship recipient

e-mail: cia.123trod@gmail.com | github: https://www.github.com/trod-123

September 2, 2018

Contents

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 that remind you 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 food assistant by speaking to your device. *Expiry Tracker* also provides additional information about how long your food can last even beyond the food's printed expiration date, so you don't throw away food that is still good to eat and save some money!

Audience

This app is for 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 can also be used by 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

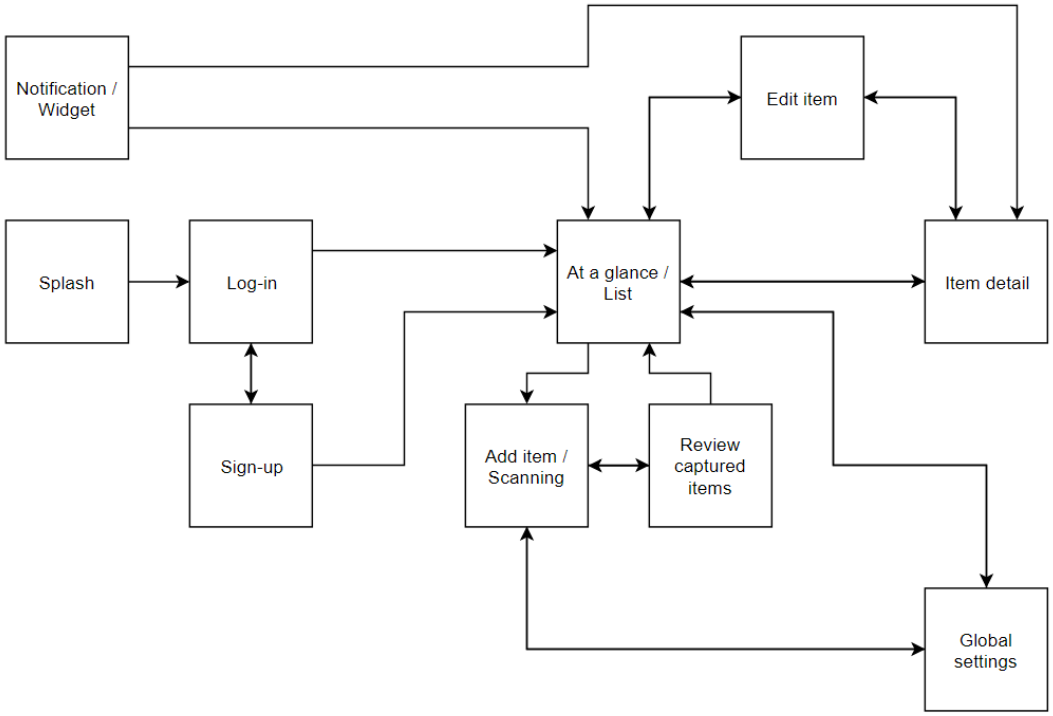
Features

- 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 and image recognition to simplify putting food into the database. Pulls your food info quickly so you don't need to enter it yourself! (requires internet connection) Also fetches images of your food online so you don't need to take pictures if you scanned a barcode (You can still add your own pictures, of course)
- Interact with your food assistant who can help store your food info for you
- Voice recognition to speed up food input, instead of typing it in
- Learn how long foods can last even beyond their printed dates
- 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

User Interface Mocks and Behaviors

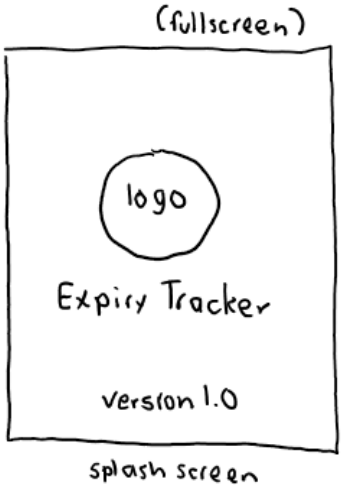
To limit the scope of this project, only mocks for phone devices on portrait mode are provided. Mocks are accompanied by thorough, yet non-exhaustive, explanations on how each screen behaves and how users interact with them. Java class names for each screen are also provided. *These descriptions and layouts may or may not reflect the app's actual final implementation*

User flow



Splash screen

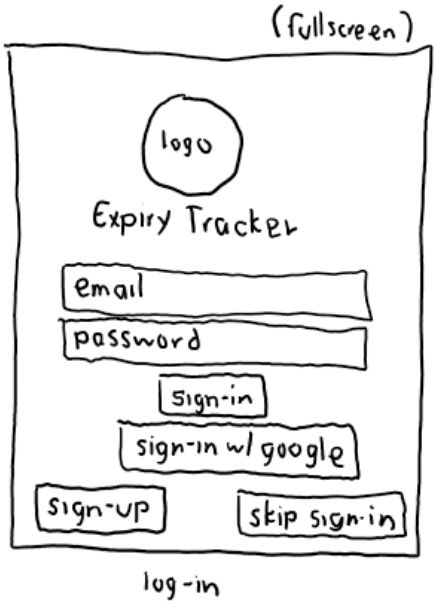
SplashActivity.java



- 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

LoginActivity.java

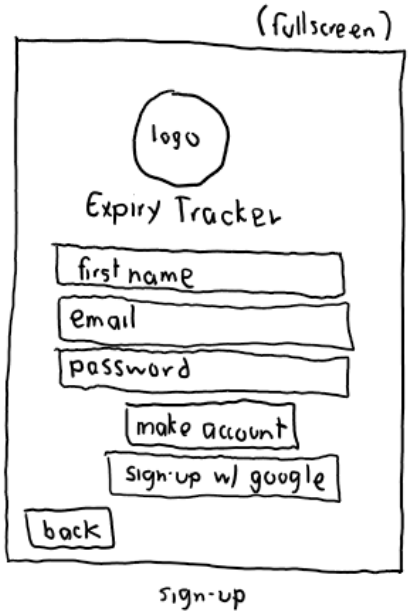


- Upon the device's first launch, users can optionally log-in to their account by entering their e-mail and password, or by "logging in with Google"
- If coming from the Splash screen, the logo moves vertically seamlessly from the center position in the Splash screen to the upper position in the Log-in 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

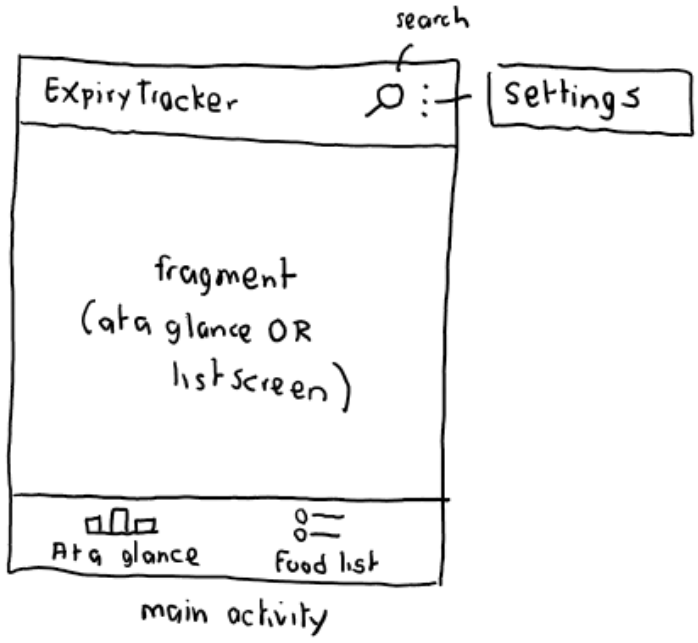
SignUpActivity.java



- 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

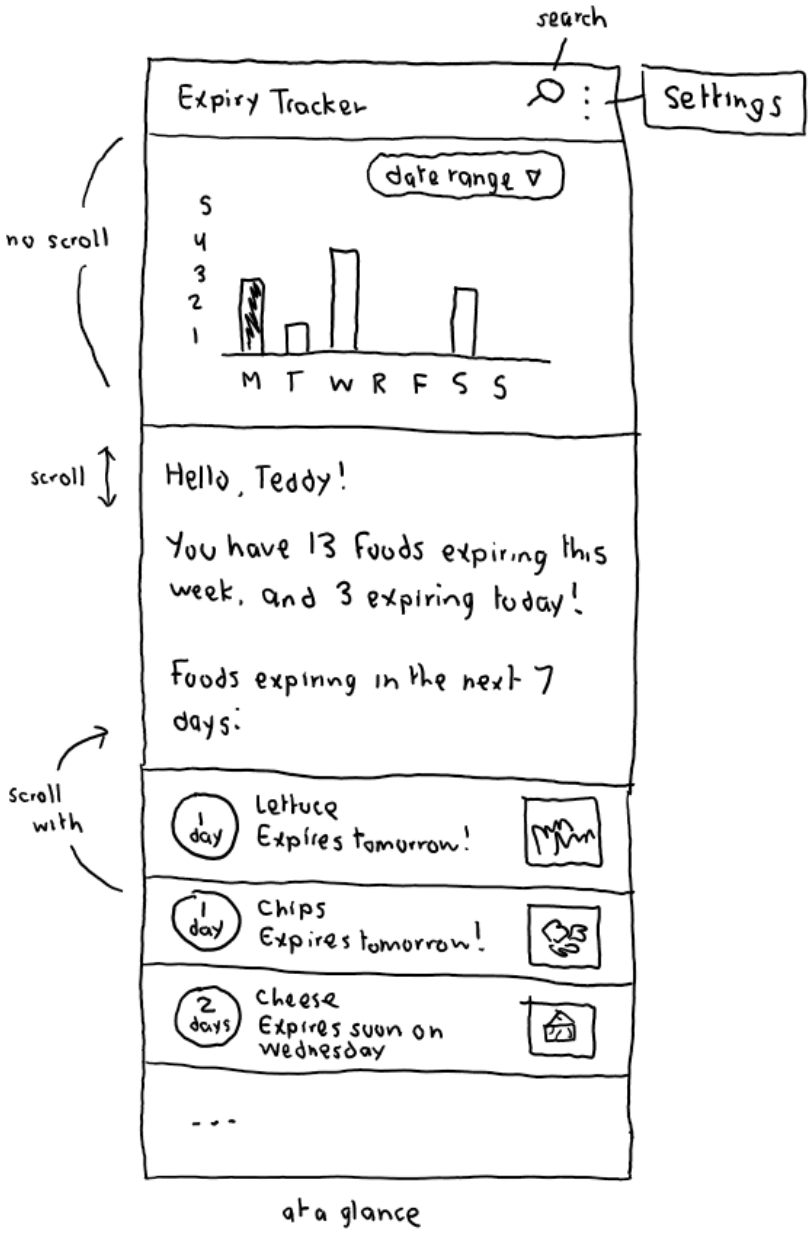
MainActivity.java



- 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)
- The Search button allows users to search for food. Implementation via AsyncTask

A. At a Glance screen

AtAGlanceFragment.java



at a glance

- 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]:

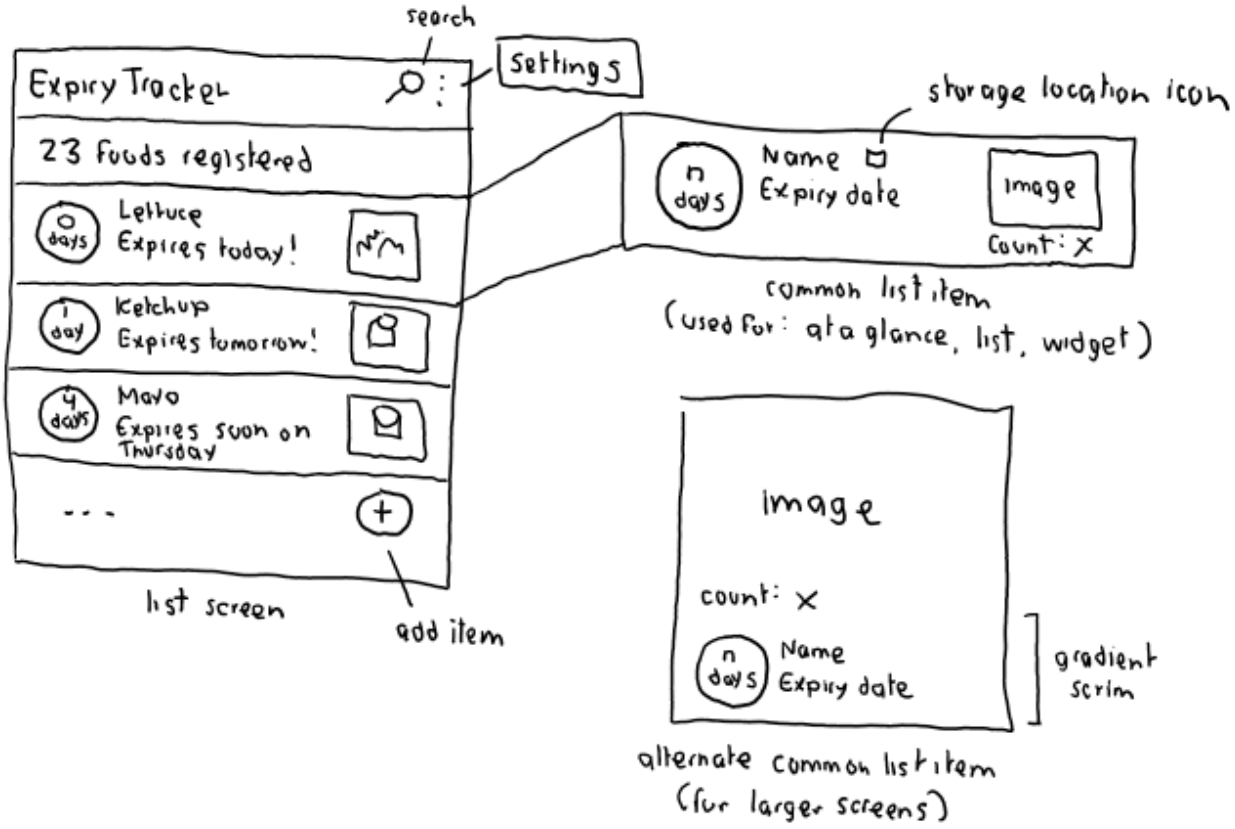
      [SHOW LIST HERE]

    - 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:
    - Vertically-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

FoodListFragment.java

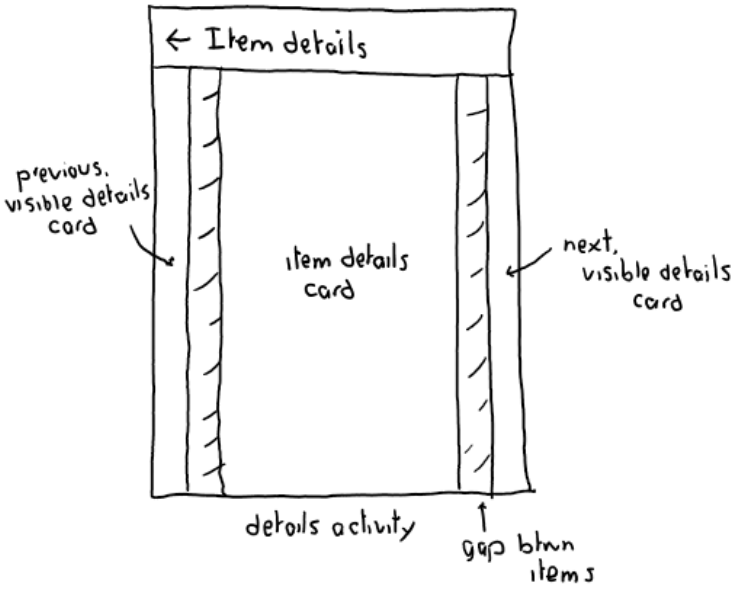


- Shows the list of all the foods users added to their database
- Above the list shows the number of foods added
- Each item contains the following views:
    - Food image, in a circular frame
    - Name
    - Expiry date, formatted based on # of days to that date
        - If it's today, formatted as "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, but titled "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

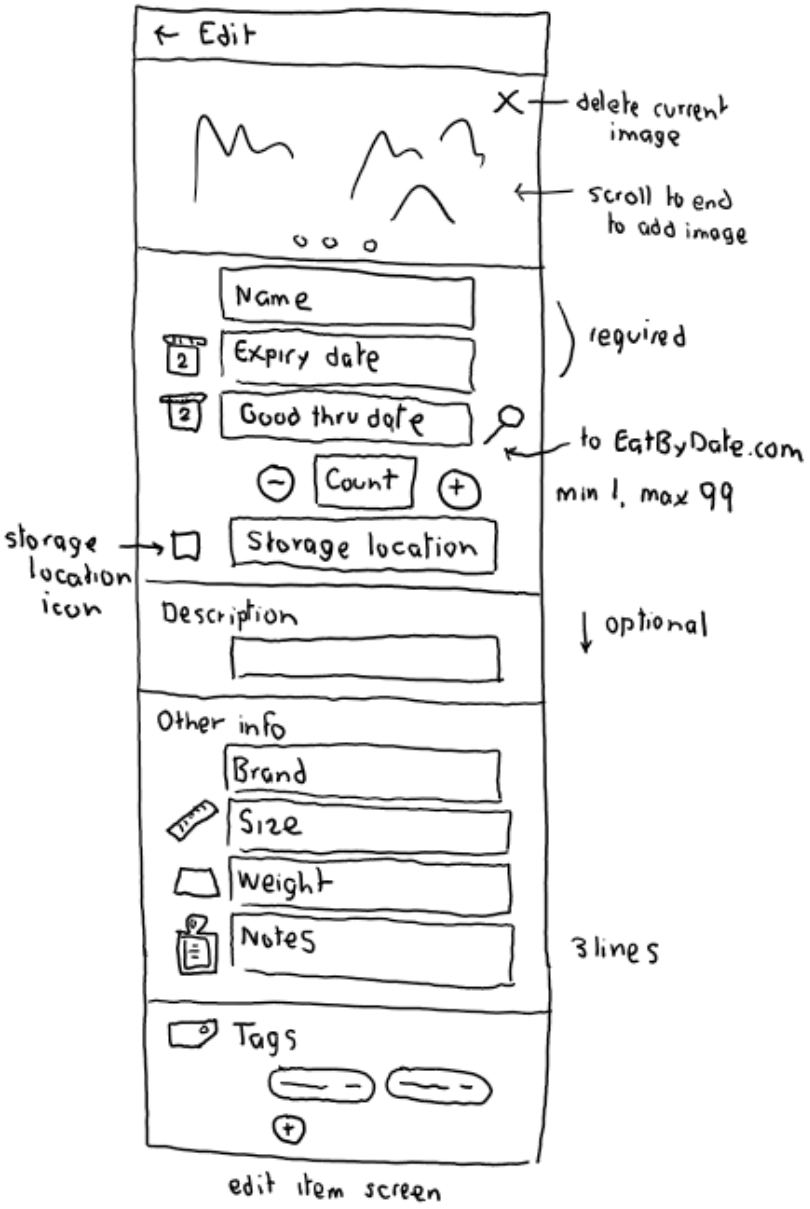
DetailActivity.java



- 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

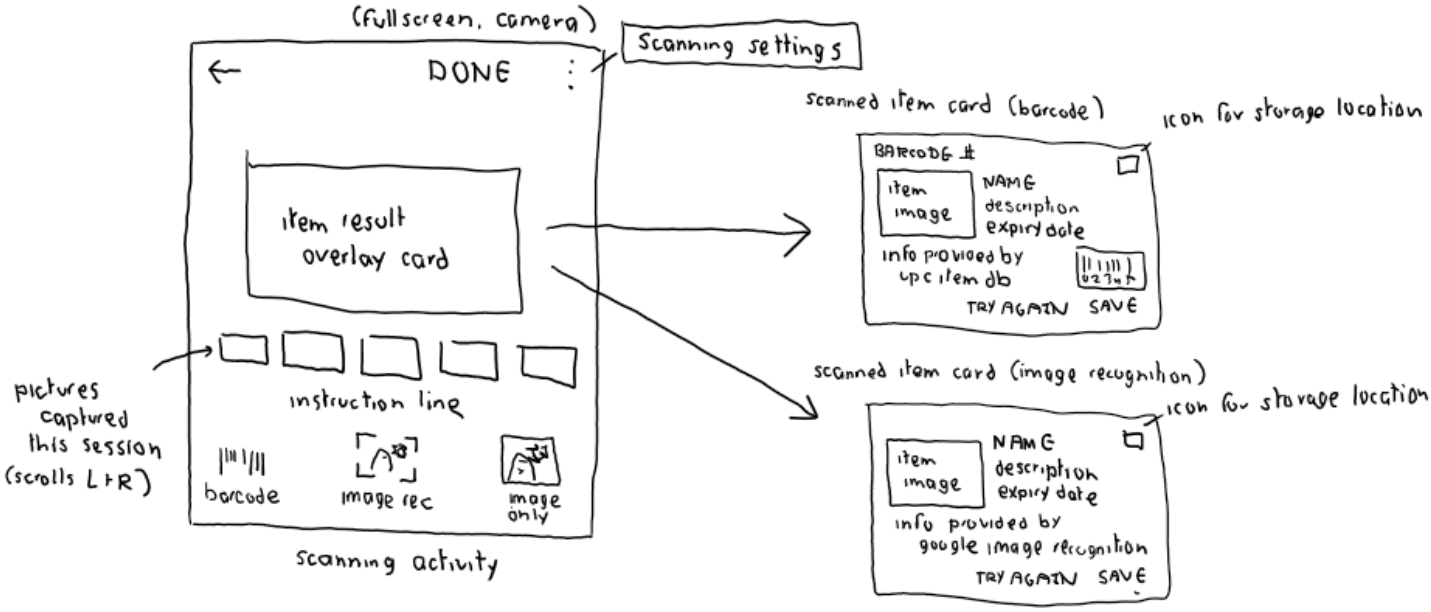
EditItemActivity.java, EditItemFragment.java



- Allows user to edit details of an item.
- User needs to press the Checkmark to save changes
- 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
    - Regardless of Capture mode, if user had already put in an Expiry date in the Add new item screen, then this date will be shown here
- By default, “Good thru date” will be the same as “Expiry date”, but users can override the “Expiry date” by entering another date for the “true” expiry date of the item (vs. the printed expiry date). Note this date must be either the same or later
- The Magnifying glass icon takes the user to a custom search page for EatByDate.com, passing in the current item name, so the user can look up the “true” expiry date for the item
    - If the name is blank, clicking the icon shows toast indicating name must be provided first
- Libraries used:
    - Glide – for image loading and caching

Add new item screen

CaptureItemActivity.java

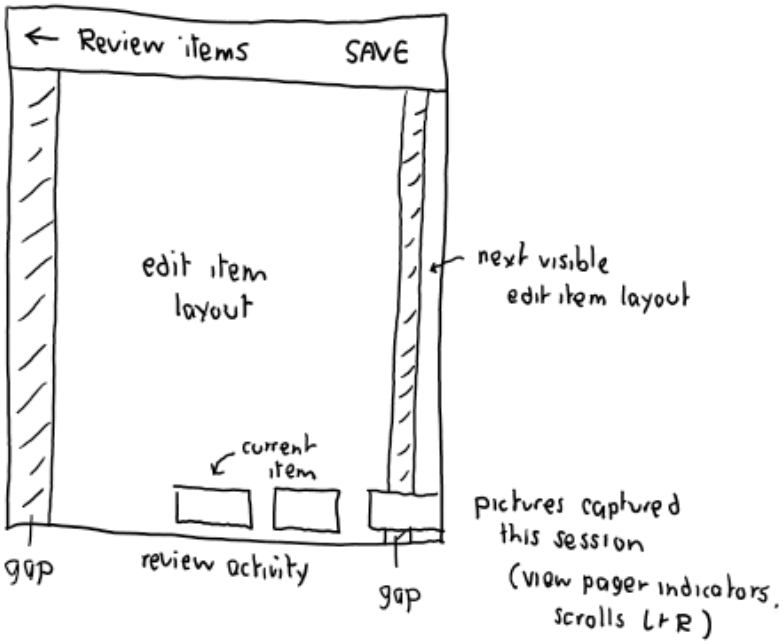


- Users are given the choice to add a new item via scanning a barcode, or taking a picture. This choice is provided via “Capture Modes” at the bottom of the screen
- The instruction line provides user instructions based on capture mode
    - Barcode: “Place the camera anywhere over a barcode to scan 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, before hitting “DONE”. Saved pictures and their associated data populate a horizontal list view above the capture mode buttons
- General scanning flow:
    - Picture taken
    - Camera continues showing, but capturing is disabled
    - 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
    - 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 (this is done by default, but users can disable this feature)
        - Name (prompt: “What is the name of this item?”)
        - Description (prompt: “Provide a brief description for this item.”)
        - Expiry date (prompt: “When does this item expire?”)

                - User is re-prompted if they say an invalid date
            - Storage location (prompt: “Where is this item stored (Freezer, Refrigerator, Pantry, Countertop)?”)
        - The app then uses Android’s Speech to Text API to handle speech to text conversions, and the card is updated with user’s input.
            - If no speech was provided, fields remain blank, and users can click on the fields to edit them
            - If users want to edit their speech input, they can still click on the fields to edit
        - 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, with images being added to the left instead of to the right
- 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: “Note: You haven’t saved your items yet, so we saved them temporarily for you so you can come back to them later.”
    - When returning to the Add item screen, these items are reloaded
- 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 captured items screen

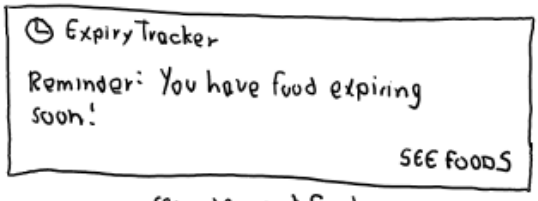
ReviewCapturedItemsActivity.java



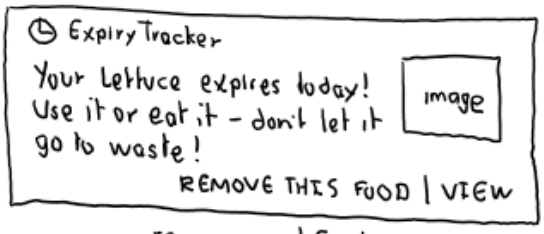
- This is where users review all the items they captured during their capture session.
- The Edit Item screen is reused here as an item in a FragmentStateViewPager. The horizontal list view on the bottom shows all the items captured, through which users can freely page through
    - The current item 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

NotificationHelper.java, NotificationJobService.java



reminder notification
multiple foods



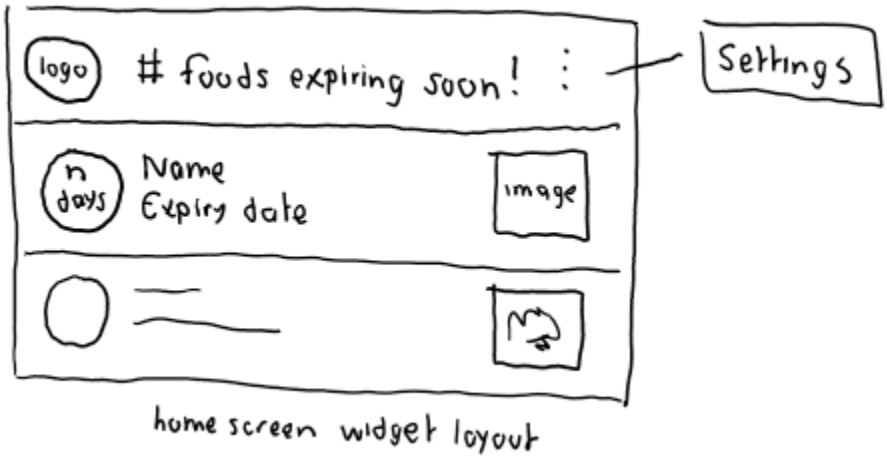
reminder notification
one food

- Users are regularly notified daily 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. Notification is dismissed afterwards
            - 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 is also shown
        - 2 buttons. Notification is dismissed after either is clicked
            - "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
- Tapping on the notification has the same effect as tapping the "See foods" or "View" buttons
- 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
- An additional notification will be provided while app is syncing with Firebase. This notification will persist while syncing, and disappear once syncing is complete

Home-screen widget

FoodWidget.java, RemoteViewsService.java, UpdateWidgetService.java



- 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)

Settings.java, preferences.xml

User settings are also saved in Firebase Realtime Database, if user has an account

- 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)
    - Enable voice input for item details (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
    - Manually sync with cloud storage (Preference; enable only if currently logged in)
    - Wipe all ExpiryTracker data on device (Preference)
    - Delete ExpiryTracker account and all data on device (Preference; enable only if currently logged in)

Key Considerations

Everything in this section is non-exhaustive and may or may not reflect the app's final implementation

Handling data persistence

- This app uses Room database to locally cache user data. To keep the UI up to date with the local database, LiveData and ViewModel are used and only when required
- This app also leverages Firebase Realtime Database and Firebase Storage to store user data and pictures online. These Firebase services allow users to download and access all of their data on any Android device, as long as they log-in with the same e-mail address they used
    - Firebase JobDispatcher is used to help manage background jobs that do the syncing with data stored online

3rd party data sources

- Food and product data submitted through barcode scanning and item search is gathered from the UPC Item Database 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 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.

Edge/Corner Cases

- General
    - In RTL layout, all horizontal ViewPagers or ListViews have their items shown in reversed order, so the "first" item becomes the "last" item. Scrolling is also reversed, so that scrolling to the left shows the next item
    - All empty lists have an accompanying "Empty" message overlay
    - All ImageViews are hidden if there is no image loaded. An error thumbnail shows if there is an error loading an image
    - "Expiry date" must be at least one day after the current day. Otherwise, throws an error and user will need to provide another date
    - Although food names and expiry dates are required to be provided, if they happen to be empty anyway, then they are shown throughout the app as:
        - Food name = "Undefined food name"
        - Expiry date = "Undefined expiry date"
- Offline/Online handling:
    - Data syncs with Firebase whenever a data action is performed (e.g. adding, editing, removing items). If we enable Firebase persistence, if the user has no internet connection when performing data actions, Firebase stores a queue of all operations that are performed, and will send those to the Firebase server when internet connection is restored

        - However, these transactions are not persisted across app restarts. To make up for this, a “cache” table in the Room database holds all the transactions that will be sent to the Firebase server, and the app will manually handle transactions this way instead, instead of letting Firebase take care of it
- At a Glance screen
    - If the food list is empty for a provided date range:
        - Bar chart is still shown, but with a message overlay: “No foods expiring this week”
        - Descriptive message greets the user and says: “You have no foods expiring this week”
        - The list of foods is hidden
    - If the user had not set a name because the user did not make an account, the descriptive message will only show a greeting without the name (and no vocative comma either)
    - If the user has no foods expiring on the current date, then the “X foods expiring today” descriptive suffix shows with a count of the number of foods expiring the next day. If there are no foods expiring then, then it looks to the earliest day with expiring foods and shows that
- Item Details card
    - If the following Detail items do not have a value set, the entire item is hidden from the view:
        - Description
        - Brand
        - Size
        - Weight
        - Notes
        - UPC
    - If all items in the “Other info” section are hidden, then this section is also hidden
    - If the food image container is empty, only the “Add image” screen shows
- Add Item screen (Scanner)
    - Pressing the back button with pictures in the horizontal list view prompts the user with an alert dialog: “Note: You haven’t saved your items yet, so we saved them temporarily for you so you can come back to them later.” (via a “Temp” table in the Room database)
        - When returning to the Add item screen, these items are reloaded
- Edit item screen / Review captured items screen
    - If coming from Add item screen:
        - Pressing the back button takes the user back to the scanning activity, if the user wants to add more items. A toast message shows: “Changes saved, but not sent to database”
            - 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
            - If there is any error associated with the Text field constraints below when the user hits “SAVE”, the app automatically pages to the first item containing the errors, and prompts the user to fix them before the user

can save. The app will do this each time the user tries to save until there are no more errors
    - If coming from Item Details screen:
        - Pressing the back button without saving will show an Alert dialog: “Your changes have not been saved” and 2 choices: Discard, Save
    - Text field constraints:
        - “Good thru date” must be either the same or later than the “Expiry date”, otherwise an error is shown
        - If the following fields aren’t filled, then an error shows when user tries to save the item. All other fields are completely optional
            - Name
            - Expiry date
        - If the user clicks the Magnifying glass without a name, then an error is shown
- Notifications
    - If the list of food is empty for an upcoming notification, then no notification will be shown

Libraries

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 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 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 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 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 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 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 15.0.1
    - For providing ads to the app's free version only
    - com.google.firebase:firebase-ads:15.0.1
- Firebase 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
- Firebase JobDispatcher 0.8.5
    - Helps send requests to sync with Firebase Realtime Database and Firebase Storage through background jobs
    - com.firebase:firebase-jobdispatcher:0.8.5
- Google Sign-in 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 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 1.6.0
    - For simplifying parsing JSON into Java objects
    - com.squareup.moshi:moshi:1.6.0
- 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 4.7.1
    - For simplifying logging and debugging
    - com.jakewharton.timber:timber:4.7.1
- 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 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 v3.0.3
    - For generating bar charts to portray numbers of expiring foods
    - com.github.PhilJay:MPAndroidChart:v3.0.3
- 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
- DiscreteScrollView 1.4.9
    - For implementing horizontal scroll view for the food items in the Add Item activity and Review captured items activity
    - com.yarolegovich:discrete-scrollview:1.4.9

APIs

Additionally, this app relies on the following APIs:

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

Implementation Notes

The following outlines and confirms that this app will fulfill the Capstone requirements provided in the **Udacity Capstone, Stage 2 – Build** rubric here.

Building and Versioning

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

The app will build from a clean repository checkout with no additional configuration. Gradle will be used to manage all app dependencies.

See below for a list of the versions of software that will be used

- 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

Libraries

See here for the list of libraries this app will use

Input Validation

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

Accessibility

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

Hardcoded values

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

Localization

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.

However, for the scope of this project, the app will be entirely in English, and no translations will be provided

Widgets

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

Google services

See here for the list of Google services this app will use. These services will be imported in the build.gradle used in this app

Ads

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

Identity

Firebase Authentication and Google Sign-in will be used to give the user a personalized experience, including addressing the user by name, and saving user data

Material design

The user interface will follow Material Design guidelines as outlined in the official documentation here, by extending AppCompat in the app theme. An app bar and its associated toolbars will also be used. Additionally, the app will have fluid animations, transitions, and shared elements to provide delightful and engaging user experiences

Flexible interface

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

Release build

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.

Data persistence

See here

Syncing data online

This app relies on FirebaseJobDispatcher to handle all data syncs with Firebase through background jobs, decoupling data transactions from the UI and ensuring data remains up to date even across the app lifecycle

Notifications

See here. Notifications will:

- Not contain advertising or content unrelated to the core function of the app
- Only be persistent when displaying Firebase sync statuses. Otherwise, they will disappear when the user dismisses them, or taps on an action button
- Be stacked into a single notification object, where possible

Developer Tasks | Pathway to App Completion

The following tasks make up a non-exhaustive set of actions needed to get this app into a working and polished state for submission. These tasks come out as a result of brainstorming and research, and may or may not reflect the app's actual implementation. Most of these tasks will be performed, but due to interest in time, some may not make it to the final submission.

Task 0: Set up the project

- Create project in Android studio
- Import all the necessary libraries
- Set the debug and release build configurations
- Set the free and paid product flavors

Task 1: Configure the App theme and set up App Settings

- Set up the custom theme
    - Choose colors for the UI
        - Primary, Secondary, and Accent colors
- Set up all of the app's settings
    - Define all of the keys, titles, and default summaries in strings.xml
    - Define all of the settings and their default values via Preferences xml
    - Implement PreferenceActivity and PreferenceFragment classes
    - Handle setting and saving all settings
    - Write Espresso tests to confirm that:
        - Settings are being saved properly
        - Correct summaries are being shown

Task 2: Prepare the UI for the Main activity, Detail activity, and Edit item activity

- Define the following resource strings (this list is non-exhaustive, make sure all strings are defined in resources):
    - app_name
    - fragment_at_a_glance_name
    - fragment_food_list_name
    - activity_details_name
    - activity_edit_item_name
    - at_a_glance_greeting_x
        - One for each of the many possible greetings (max 10)
        - Include a morning, afternoon, and evening greeting (max 2 each)
    - days_of_week_labels (array)
        - String array, with one string for each day of the week

- at_a_glance_bar_chart_label
- at_a_glance_bar_chart_empty
- at_a_glance_date_range_label
- at_a_glance_date_range_values (array)
    - String array, with one string for each accepted date range values
- at_a_glance_summary
- at_a_glance_summary_postfix
- at_a_glance_summary_empty
- at_a_glance_list_header
- action_search
- action_settings
- action_add_item
- action_add_item_camera
- action_add_item_image
- action_add_item_text
- action_save
- expiry_msg_today
- expiry_msg_tomorrow
- expiry_msg_soon
- expiry_msg_next_week
- expiry_msg_ordinal
- expiry_msg_month_day
- list_empty
- list_registered_header
- food_name_label
- food_name_none
- food_expiry_date_label
- food_expiry_date_none
- food_good_thru_date_label
- food_days_label (plural)
- food_storage_location_label
- food_count_label
- food_description_label
- food_other_info_label
- food_brand_label
- food_size_label
- food_weight_label
- food_notes_label
- food_meta_data_label
- food_barcode_label
- food_tags_label
- food_capture_label
- food_capture_barcode_label
- food_capture_imgrec_label

    - food_capture_imgonly_label
    - food_capture_textonly_label
    - data_attribution_upcitemdb
    - data_attribution_google_imgrec
- Create or import the following icons:
    - ic_action_search
    - ic_action_add
    - ic_action_edit
    - ic_action_delete
    - ic_storage_freezer
    - ic_storage_refrigerator
    - ic_storage_pantry
    - ic_storage_countertop
    - ic_storage_custom
    - ic_calendar
    - ic_ruler
    - ic_weight
    - ic_clipboard
    - ic_tag
    - ic_show
    - ic_hide
- Choose colors for the UI:
    - expires_today
    - expires_tomorrow
    - expires_following
    - expires_default
    - list_item_delete_background
    - icon_tint
- Create the following layout XMLs:
    - activity_main.xml
    - fragment_at_a_glance.xml
    - item_food.xml
        - layout_common_food.xml
        - layout_common_food_alt.xml
        - layout_number_circle.xml
    - fragment_food_list.xml
    - activity_details.xml
    - fragment_item_details.xml
    - acitivty_edit_item.xml
    - fragment_edit_item.xml
- Create a helper class with the following methods:
    - generateRandomGreeting()
    - generateSummary()
    - generateBarChartXAxisValues()

    - generateExpiryString()
    - showToast()
- Create Java classes for each of these activities and their associated fragments
    - Create Adapters for all lists
    - Create FragmentPagerAdapters for MainActivity and DetailActivity
    - Feed dummy data into the UI
    - Define all button behaviors
- Create action menus for MainActivity and EditItemActivity
    - Make sure all actions and labels are defined in strings.xml
    - Define the following XMLs
        - menu_main.xml
        - menu_edit_item.xml
    - Inflate menus and handle actions in their Java classes
- Write tests to verify that settings values are being properly read for the following:
    - Display name (AtAGlanceFragment)

Task 3: Prepare the UI for the Capture Item activity and the Review Captured Items activity

- Define the following resource strings (this list is non-exhaustive, make sure all strings are defined in resources):
    - capture_barcode_label
    - capture_imgrec_label
    - capture_imgonly_label
    - capture_instruction_barcode
    - capture_instruction_img
    - action_done
    - voice_prompt_name
    - voice_prompt_description
    - voice_prompt_expiry_date
    - voice_prompt_storage_location
- Create the following layout XMLs:
    - activity_item_capture.xml (full screen camera layout)
        - card_scanned_item_barcode.xml
        - card_scanned_item_imgrec.xml
        - card_scanned_item_imgonly.xml
        - layout_horizontal_image_scrollview.xml
        - item_image_scrollview_preview.xml
    - activity_review_captured_items.xml
- Create or import the following icons:
    - ic_capture_barcode
    - ic_capture_imgrec
    - ic_capture_imgonly
- Create Java classes for each of these activities and their associated fragments

    - Create Adapters for the horizontal image scrollviews used in CaptureItemActivity and ReviewCapturedItemsActivity
    - Feed dummy data into the UI
    - Define all button behaviors
- Create action menus for CaptureItemActivity and ReviewCapturedItemsActivity
    - Make sure all actions and labels are defined in strings.xml
    - Define the following XMLs
        - menu_item_capture.xml
        - menu_review_captured_items.xml
    - Inflate menus and handle actions in their Java classes
- Create entire scanning flow for capturing an image, and feeding it data
    - Implement getting API response from UPCItemDb, Google Image Recognition, and Speech Input features (don't make Food objects yet at this point, just return the JSON response string)
    - Write tests to verify that UPCItemDb data is being returned
    - Handle attempts to fetch data from UPCItemDb when offline
    - Store all temp data in a dummy cache when sending between CaptureItemActivity and ReviewCapturedItemsActivity
        - Write tests to verify that data is sent properly between the activities (including sending to ReviewCapturedItemsActivity, making edits there, and sending it back to CaptureItemActivity)
- Write tests to verify that settings values are being properly read for the following:
    - All capture setting

Task 4: Create the on-device food storage database

- Define the tables and each of their schema
    - Food table – this makes up the "real" database, which is synced with Firebase
    - Temp table – this is a hold for items captured, but not yet saved *(not to be used just yet for this task)*
    - Cache table – this is a hold to keep track of database transactions that were done offline or when connections could not be made with Firebase
- Implement Room database for each of the above tables and schema (schema should be the same across all 3 tables)
    - Entities
    - DAOs with LiveData
    - RoomDatabase subclass
    - Repository class
- Parse UPC Item Database and Google Image Recognition API responses into Food objects
- Populate the database with dummy data
- Write tests to verify that API data parsing is done correctly
- Write tests to verify that data is being written and read from the database correctly

Task 5: Link the on-device food storage database to the UI

- Create ViewHolders and link them to UI fragments
- Implement Paging library for RecyclerViews
- Connect LiveData to UI via Observers
- Write tests to verify that dummy data is being loaded to the UI properly
- Integrate the “Temp” Room table to the CaptureItemActivity and ReviewCapturedItemsActivity

Task 6: Set up user accounts and LoginActivity and SignUpActivity

- Set up Firebase Auth
- Connect the app to Firebase
- Define the following resource strings (this list is non-exhaustive, make sure all strings are defined in resources):
    - auth_email_label
    - auth_password_label
    - auth_password_rules
    - auth_login_label
    - auth_login_error_no_internet
    - auth_login_error_no_account_found
    - auth_login_error_blank_fields
    - auth_signup_label
    - auth_skip_signin_label
    - auth_name_label
    - auth_make_account_label
    - auth_make_account_error_blank_fields
    - auth_back_label
- Create layouts and associated Java classes
    - activity_login.xml
    - activity_signup.xml
- Write tests to verify that logging in and making a new account writes to the appropriate settings
    - Display name
    - Sign-out button (this should be shown instead of the Sign-in button)
    - Delete ExpiryTracker account (this button should be enabled)

Task 7: Create the Firebase Realtime Database and Firebase Storage

- Set up Firebase Realtime Database and Firebase Storage
- Connect the app to Firebase
- Create a FirebaseJobService that extends JobService, which would be in charge of reading and writing data to and from Firebase
- Write tests to verify that data is sent between the app and Firebase properly, using FirebaseJobDispatcher to schedule and run FirebaseJobService

Task 8: Link on-device data with Firebase

- Let the Repository interface Room with Firebase. Use FirebaseJobDispatcher to schedule and run FirebaseJobService that handles data transactions with Firebase
    - This should also be where Firebase Auth does its magic to link with Firebase Realtime Database and Firebase Storage
    - Use the “Cache” Room table to store all items that will be sent to Firebase. In this way, if the user does not have internet connection, this data will still be able to sync up with Firebase once connection is re-established
        - How this works: Data that is sent to the “Food” table is also sent to the “Cache” table. Data from the “Cache” table is sent to Firebase, and if operation is successful, is deleted from the “Cache” table. Otherwise, it remains in the “Cache” table for the next sync attempt
    - When downloading from Firebase, data is sent straight to the “Food” table
- Sync user settings with Firebase:
    - All Notification settings
    - All Widget settings
    - All Capture settings
    - Display name
- Write tests to verify that the “Cache” table sends data to Firebase only when the user has internet connection, and that the data is deleted afterwards
- Write tests to verify that the “Cache” table fails to send data to Firebase when there is no internet connection, and that the data remains in the table
- Write tests to verify that new data from Firebase is delivered to the “Food” table

Task 9: Configure Notifications and Widgets

- Create the notifications and their pertinent actions in a NotificationHelper class:
    - Multiple foods notification
    - Single food notification
    - Firebase sync status (a different, persistent notification that shows only while the app is syncing with Firebase – should not be impacted by user-provided notification settings)
- Use a JobScheduler to run a NotificationJobService at intervals that the user provides in settings:
    - Receive reminder notifications?
    - Be reminded when?
    - Time of day to be reminded?
- Verify that notifications are showing properly, based on the intervals in settings
- Create the Widget configuration and the Widget layout XML
    - widget_info.xml
    - widget_layout.xml
    - widget_item.xml
- Implement the following Java classes to program the Widget:
    - RemoteViewsService.java
    - FoodWidget.java
    - UpdateWidgetService.java
- Verify that the widget provides the correct list of foods

Task 10: Create the Splash screen and design the App logo and notification icon

- Define the following resource strings (this list is non-exhaustive, make sure all strings are defined in resources):
    - app_version_number
- Create layout and Java class for SplashActivity
    - activity_splash.xml
- Create the logo and import it into the app into the following places:
    - SplashActivity
    - LoginActivity
    - SignUpActivity
    - Widget
- Create the notification icon and import it into the app in the following places:
    - Notification

Task 11: Connect EditItemActivity with EatByDate custom search

- Create layout and Java class for EatByDateWebViewActivity
    - activity_eatbydate_webview.xml
- Verify that the back button returns to the EditItemActivity
- Handle offline edge cases
    - Show an error if user taps on Magnifying glass in EditItemActivity when there is no internet connection
    - Show an error in EatByDateWebViewActivity if there is no internet connection

Task 12: Optimize layouts for different device sizes and configurations

- Create optimized layouts for landscape mode and tablet mode
    - Use Multi-Pane layouts and Compound views for tablet mode where possible
- Verify that layouts are showing fine across different devices

Task 13: Polish the app with animations and transitions

- Add transitions to move the logo from SplashActivity to its place in LoginActivity, if the user has to log-in
- Add continuous shared elements transitions between List and Detail screens
- Add appropriate enter and exit transitions when traversing screens

Task 14: Implement ads for the Free version only

- Confirm the Firebase Mobile Ads SDK has been imported
- Register and link this app to a Firebase project
- Get the AdMob app ID and use it to initialize the Mobile Ads SDK when the app launches

- Add banner ads only for the following layouts:
    - MainActiivty: above the tabs
    - DetailsActivity: on the bottom
- Add an interstitial ad after saving items in the ReviewCapturedItemsActivity, before returning to the MainActivity
- Verify that ads are only showing for the Free version, and no ads are showing for the Paid version

Task 15: Build the Release version of the app

- Ensure the keystore and password with the signing configuration properly generate a signed, release apk

Task 16: Run final testing for the Release Free and Release Paid versions of the app

- Verify that all tests made in the previous tasks pass
- Verify that the app does not crash anywhere
- Allow friends and family to test the app to ensure the experience is as intended
- Verify the app is good for submission!

Task 17: Submit!