MSD Capstone Project Final Report:

CrispList

Andrew Kellett

CS 6019

Dec 7th, 2022

Table of Contents

Introduction

In the United States, the Food and Drug Administration estimates that between 30-40% of our food supply ends up as waste. “Wasted food is the single largest category of material placed in municipal landfills and represents nourishment that could have helped feed families in need.” [1] This contributes to food scarcity. Food waste also represents an impact on the environment. Globally, food waste emissions are nearly the same as global road transportation emissions.[2] The energy, water, and labor used to produce that wasted food could have been applied effectively elsewhere.

The federal government of the United States is attempting to rectify this by educating its citizens on how to reduce food waste and loss. Some of the solutions that its education program suggests include consumer-level advice, such as planning meals, learning how to buy and store foods safely, and keeping track of what needs to be used before it becomes waste. While it may not solve all the above issues, I think a mobile

app could be instrumental in in helping its users save money, reduce waste, improve health, and be better stewards of the environment at the consumer level.

In my own family of 5 people, we have experienced food waste first-hand. Having three children, our household food consumption has increased, so buying more food (and in bulk) has become necessary. Our family refrigerator at times is crowded with food after large grocery trips. Often, items of produce or leftover meals get pushed to the back of the shelf or buried in the produce drawer where they are easily forgotten and out of view. Upon eventual re-discovery of the items, they are often moldy or spoiled and must be thrown away. This recurring experience served as the main inspiration for this project and we intend to use this app in our own home.

Background

I set out to make an app that acts as a meal planner, a grocery list, and an expiration date tracker. From what I could tell, current app offerings are either an expiry tracker or a grocery/meal planning app – not both. I didn't see any solutions that combine the functions of planning, shopping, and tracking foods users have purchased. In the context of reducing food waste and saving money, the need for one app to manage all aspects of planning, shopping, and using the food effectively became clear.

In order to ensure that the app would be effective in its purpose, I decided that simplicity would need to be paramount in design. Implementing features that enabled multiple users in the household to use the app together is important. A whole-home or family approach to plan and reduce waste would seem more effective as a group endeavor. I wanted to enable list editing by multiple parties and have updates render live on grocery lists, much like a google doc.

Solution Description

Tools

To build out this app and to make it available across several platforms, I used the Flutter framework which uses the Dart language. Flutter is an effective way to build user interfaces and logic that result in an app for iOS, Android, and the web. Flutter gave my project more accessibility to devices of all types and eliminated the need for separate codebases for each platform. The Dart language enabled me to build quickly and see the product develop instantly in the emulator with its ‘hot reload’ feature. Also, dart is declarative, which was useful for having widgets and UI change according to the state changes of the data in the database. This meant I could make a change in the database and see the UI change instantly per the listeners.

The app required a robust backend that would have to be responsive and able to update while multiple users view and edit shared data. For this reason, I decided not to make my own homemade backend solution but to use the well-established Google Firebase. With the use of its products like Auth and Firestore, I incorporated authorization and a place to store app data.

Features

- Each feature’s data is backed up by google firestore and the state of the data is set up on streams. If a list is changed, an ingredient deleted from a recipe, or if

someone changes an expiration date, the database and all devices pointing to that data will reflect that change instantly.

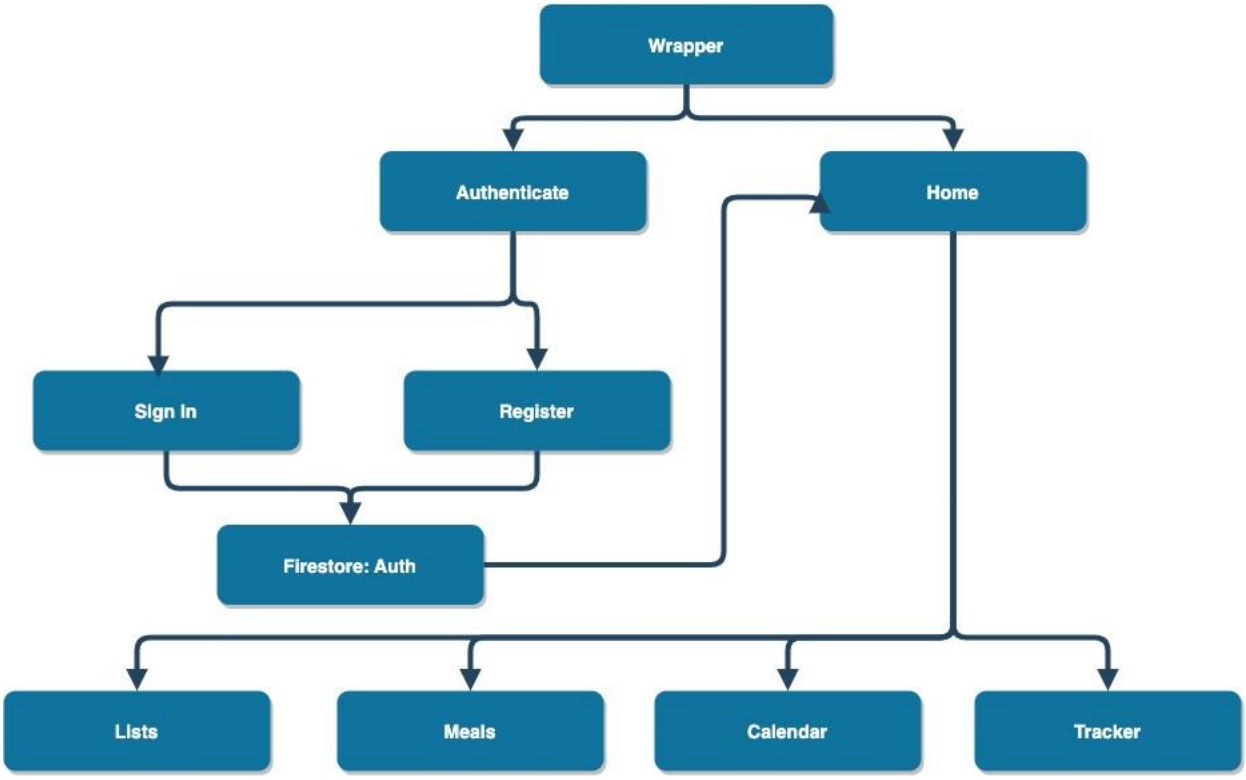
- A page where the user can make a grocery list, sorted by user-specified categories (dairy, produce, packaged foods, etc.) generally mirroring aisles/departments of grocery stores. The rationale behind this choice is that a user won't have to re-visit, say, the cereal aisle if user was following the grocery list in order and the two cereal items were on opposite ends of the list. The list consists of tiles that can be checked on/off, the name of the item and the quantity. As required, each list item also has a detail view to specify the item's name, quantity, package size, any notes the user wants to include, and the store at which the user plans to buy the item (i.e.: "Costco" vs "Walmart"). The item's detail view is accessed by dragging the tile to the left side of the screen and selecting the detail button. If the user swipes the tile in the other direction, an option to delete the grocery list item appears. If the item's category is changed, the list tiles will reorder according to those changes. An add button below brings up a drawer containing a form to add new items to the list.
- A page where the user can enter their meal recipes and another page where they can plan those meals on a calendar. As people typically repeat the meals they eat, meal recipes only need to be added to the app only once. Then the user can assign it to however many days they wish. From the meal page, which hosts slidable list tiles, users can swipe to delete recipes or view a screen listing the recipe ingredients. Buttons surrounding each ingredient either let one view/edit ingredient details on a detail screen, or they can add their selected

recipe ingredients to the shopping list as needed with a click of a button. The calendar is still being implemented to reflect the meals assigned to dates. However, the calendar is currently implemented and is also interactive—the calendar UI responds to selection and its view can be toggled to different configurations (one week, month, two week, etc).

- A page where a user keeps track of the food they purchased. On this page, the user can assign applicable expiration dates to remind themselves to use up specified foods prior to expiration. The list of tiles is sorted by the date the items expire. The top of the list showing the item that has expired or is the closest to expiration. As a visual cue to the user, Items that have expired are a deep red and say “EXPIRED!”, Items within 2 days of expiring are a less deep red and display “LAST DAY!” on them if the current date matches the expiration date. For items 3-4 days from expiry, the tile is orange. Items 5-7 days left to expiry are colored yellow. Not only does this create a pleasant sunset of warm colors, it also helps remind users of food/leftovers—alerting them to what foods need to be prioritized. Item names and expiration dates can be adjusted in detail panels that appear when the item is dragged to the left and the button selected. The tracking feature also helps users away from their home to know if they’ve already purchased an item (i.e.- buying another sour cream at the store because you couldn’t remember if you had any at home). When the item has been used up or thrown away, users swipe to delete the item from the list. As users add, edit, or delete list items, the list is reordered accordingly.

- Each of the four main features are navigable via an animated navigation bar found at the bottom of the device screen.

Widget Flow and Classes



The above diagram depicts how the main widgets flow, one to another. Upon app launch, the wrapper widget detects if a user is signed in and sends the user to the home widget if logged in or to the Authenticate widget if not. The Sign in or register widgets are where the user actually can sign in/sign up, after which, the app communicates with Firebase to confirm the user's login was valid. If successful, the data is retrieved and populates the appropriate widgets. The home screen by default is the grocery list page.

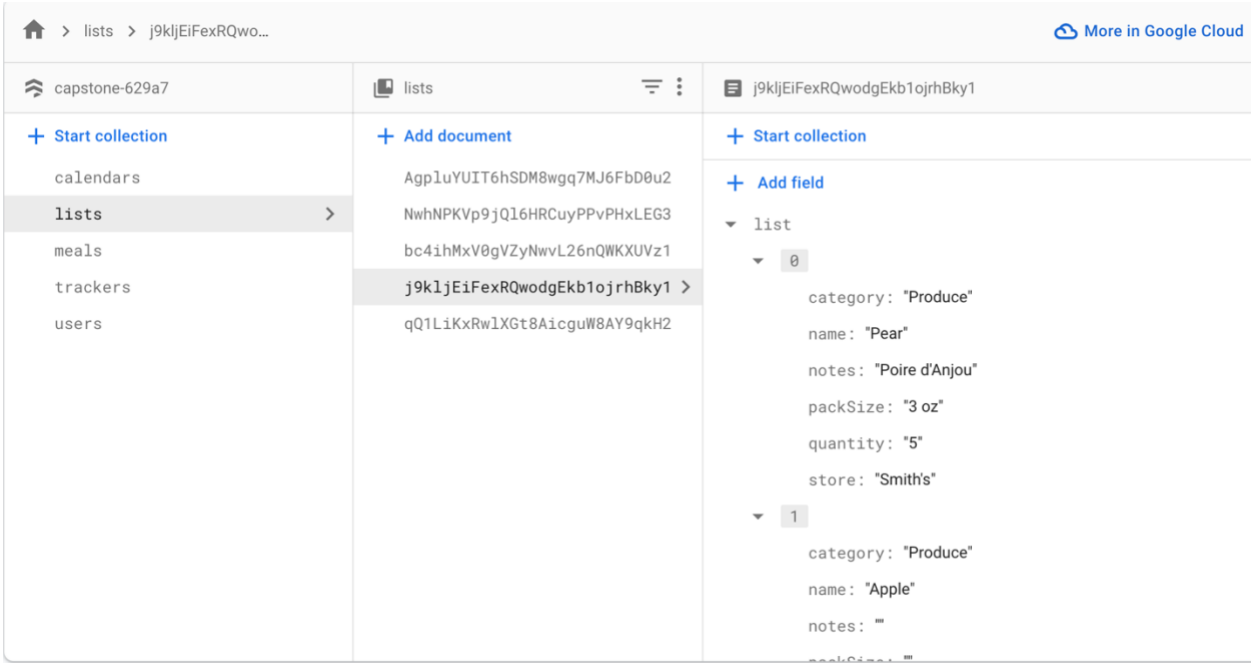
This was a choice made with the user in mind—when the user pulls out their phone at the store, they are likely to want the grocery list to pop up first thing rather than having to hunt through the app for it, increasing time at the store each time the app restarts or is shut down. Users can navigate to the other pages via routes connected to the nav bar.

Some of the widget classes are for UI purposes while others serve as either helper or data classes. One of the most integral classes I constructed is the Database Service class/widget. It handles the incoming/outgoing communication of the app with the firestore cloud storage. That class calls for the conversion of data back and forth between Firestore-storable data types and my custom data objects and vice versa. It is also the class that is responsible for data operations like creating, reading, updating, and deleting items in the database.

Another important widget class is that which handles the login/registration. It communicates directly with Firebase's Auth service. Auth works very well as I'm able to let the service handle user/password storage, unique identifiers, and handling operations like forgotten passwords and verification. On the client side, I use a few data verifiers to make sure the passwords and emails entered in are valid/in proper format.

In order to have changes to the data in the database push changes to the data in each user's app, I use the provider class and other related listener objects. Once the state is changed in the DB, the changes push to the device via streams to the proper widgets and a UI "redraw" takes place with the updated information.

Database Organization



Firestore is a NoSQL database organized into collections of documents (see the above image). I've organized the database into several collections (i.e.- lists, calendars, etc). Each document in the 'lists' collection, for example, represents a user's complete grocery list. The list is an array of nested JSON-like objects that have fields for name, quantity, etc. Firestore already gives each document a unique key by which it is referenced. The other collections are of similar structure with key value pairs.

Results

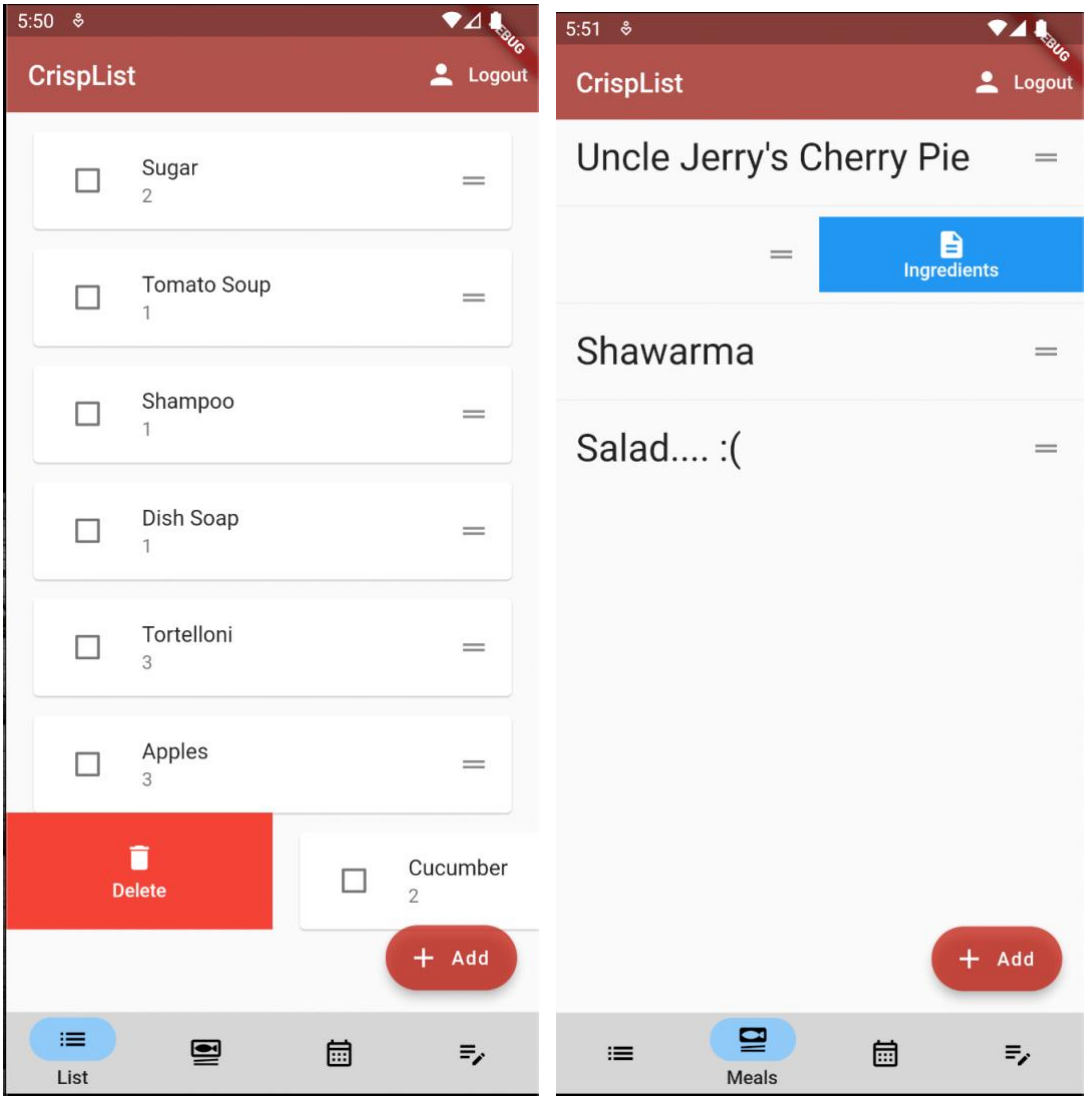
The app is functional and communicates well with the database. In implementing various C.R.U.D. functions, I successfully debugged and worked out asynchronous details to improve performance. This was one of the most time-consuming aspects of the project. The app data updates with changes very consistently and is responsive – the UI doesn't really get hung up loading. I find it useful and usable despite a few bugs. My favorite aspect of the app is the expiration tracker page. It's functional and I love the colors. What I've accomplished so far makes me want to work on it more!

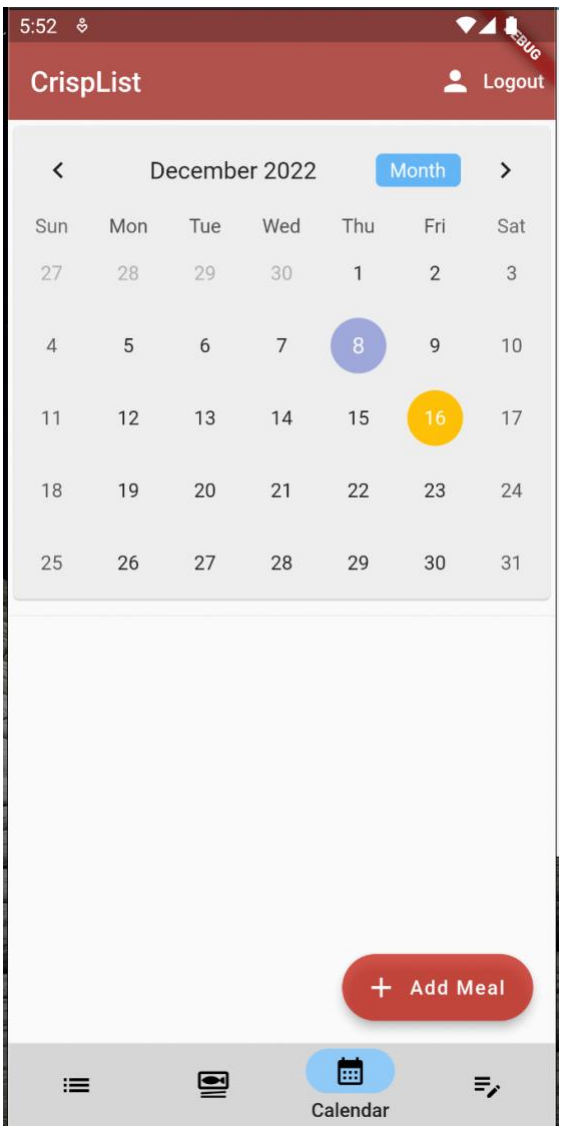
There are still aspects of the project that I had planned to finish but did not get finished within the course of the semester. The app is functional, but I plan on continuing to work on it as I have time in the future in order to finish implementing my planned features. I'd like to implement more on the calendar page to show the meals assigned to the selected dates. Next, I would like to finish enabling the sharing of grocery lists between users beyond the preliminary framework. Yet another goal would be to display actual expiration dates and not just colors on the expiration list. Lastly, I would work harder to decorate and polish the UI design.

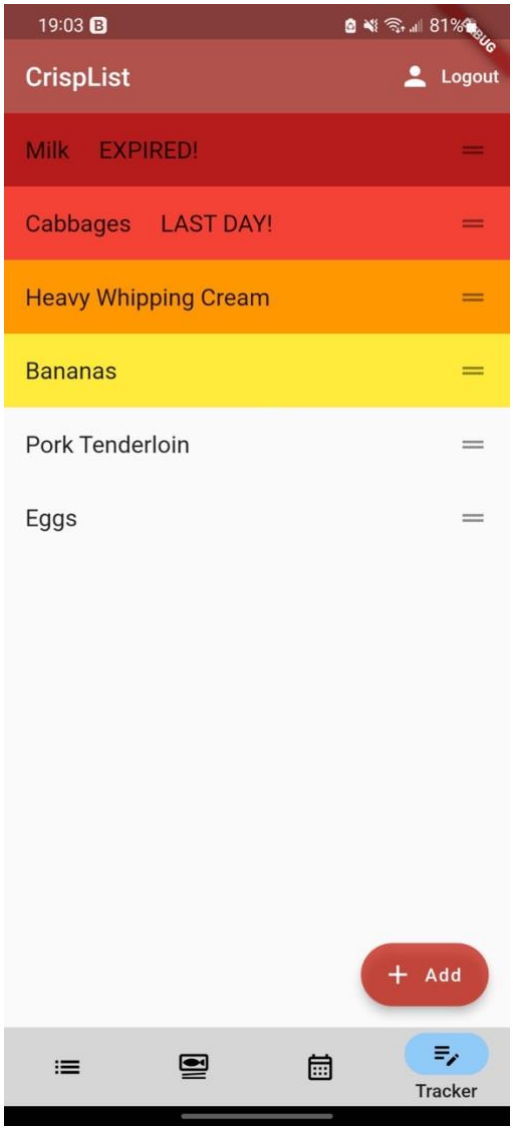
One of the major limitations of the project is its current reliance upon the internet to make changes to the local UI. The frameworks do have an aspect of persistence. For example, if I turn off internet to the device and then open the app, it will still contain all the lists, recipes, expiration date items from the last time the app opened. The limitation comes where I want to take that cached list and make a change (delete, edit, or create an item)—the app freezes and no changes are made. This is due to the current flow of data. When a device makes a change, the database is fed the changes, which then

comes back to notify the stream in-app that it needs to update. If lack of internet severs that loop, it cannot update its own cached data, it can only display it.

Screenshots







Conclusions

Lessons Learned

Besides learning a new language and framework, I learned that when building a project, more time should be built in the plan for debugging and testing. During my integrating of the database with the main app, I experienced errors that took me almost a week and a half to fix. Sometimes, even though we try to plan for setbacks, the

progress timetable gets pushed back. Due to a family emergency, for example, I got behind by at least a week in my time table. In the context of future work, I feel like I'll be more cognizant of those types of setbacks and will try to account for them in my timeframe estimations, especially where new skills are being learned.

This project taught me even more about how to plan, design, and execute a project even if it's dealing with a new language and framework that I have to learn on my own. Another interesting perspective I gained was from using Flutter itself: I learned the differences in declarative style UI and imperative style. Overall, this project gave me a greater confidence as it required me to teach myself what I was lacking without any help from others. It was really neat to see a project forming as I added to it. I learned so much!

Future goals for the project

I have a list of additional goals to strive for, circumstances permitting. I'd like to implement a few other features that I think would be useful and contribute richly to the user's experience.

- Foodsafety.gov has a web app called FoodKeeper that teaches people about how to store their food-- especially perishables—so as to extend freshness and provide expiry guidelines for fresh items that don't have a factory-assigned expiration date. Somehow implementing this app or its info would be useful information for the user to reference in the app. If I cannot find a way to access it successfully in the app, I would include it on a page of external resource links.

- For ease of use, I'd like to implement barcode scanning of purchased items. Flutter can access a kotlin plugin called "barcode_scan." For this feature, the user could scan the barcode of a bag of frozen peas or a can of tuna and have the app add the product information to the list of user's food (Tuna, 5oz). This way, the user only needs to enter in the expiration date manually. Admittedly, I would need to find and access a barcode database, or just have it be user populated via manual entries of all users—a communal effort. Also, I may look into having the mobile camera scan the actual expiration date and enter it into the appropriate field.

Works Cited

[1]

Center for Food Safety and Applied Nutrition, "Food loss and waste," *U.S. Food and Drug Administration*. [Online]. Available: https://www.fda.gov/food/consumers/food-loss-and-waste. [Accessed: 08-Dec-2022].

[2]

"Food wastage footprint & Climate Change." [Online]. Available: https://www.fao.org/3/bb144e/bb144e.pdf