SP-5 Red Dynamic Grocery List App

Final Report

CS 4850 – Section 02 – Spring 2024

April 27, 2024

Team Members:

Roles	Name	Major responsibilities	Cell Phone / Alt Email
Team Leader / Documentation	Aaron Figueroa	Facilitates communications between advisor and team and composes deliverables.	646-709-3919 adfig13@gmail.com
Documentation	Elizabeth Krummert	Composes deliverables for the project.	770-893-8886 ekrummert8814@gmail.com
Developer	Cole Young	Develop and maintain core functionality of application.	706-581-0894 coleturneryoung@gmail.com
Advisor / Instructor	Sharon Perry	Facilitate project progress; advise on project planning and management.	770-329-3895

Website Link: https://sp-05red.github.io/

Project GitHub: https://github.com/SP-05Red

Number of Lines of Code: 950

Number of Components: 8

Total Man Hours: 142

Table of Contents

1.0 Introduction

The application GrocAgree is an easy-to-use application that allows users to create and share grocery lists with other users. To develop this application, we used Flutter to develop the UI and Dart as the programming language for the application. The backend for the app was managed using Google Firebase. We used Firebase Authentication to manage user login and sign-up features. We also used Firestore Database to store the data generated by users. The initial layout of the app's design was created using Figma, which allowed us to get an idea of how the app was supposed to look and function before development began. Version control of the app was managed using GitHub and we used Visual Studio Code as the IDE for development.

1.1 Definitions and Acronyms

UI: User Interface

UX: User Experience

UID: Unique ID

SRS: Software Requirements Specification

VS Code: Visual Studio Code

2.0 Planning

2.1 Technologies Used

For our project, we decided on technologies that would best fit our blend of meeting project efficiency and user's needs/experience. The core of our development process was done using Visual Studio Code, as our primary integrated development environment (IDE). We chose Flutter and Dart for our cross-platform application, due to the team's experience and to ensure a great user interface across IOS and Android.

For online meetings and communications, we utilized Microsoft Teams throughout the project's development. We communicated and shared files throughout development as well on Teams.

We utilized Firebase Authentication and Firebase Firestore to ensure that all data being handled in our application is handled in real-time. We utilized collections in Firestore that handled the lists and their fields that are generated by our application. Firebase and its numerous libraries for various languages helped greatly simplify connecting our application to its real-time databases. Another reason we chose Firebase is that, like Flutter and Dart, it was developed by Google and ensured easy and seamless cooperation between the app and database.

For our version control and team collaboration, we utilized GitHub for our development workflow. We were able to use the GitHub Codespaces within VS Code for direct collaboration with the latest versions of our application.

Lastly, Figma played a key role in our design process. It allowed us to craft models and representations of our UI and UX flow, as well as create a vision for the user interface we wanted to create to be able to give our user's the best experience.

2.2 Collaboration and Communication

We decided to meet twice a week to discuss the project, once in person and once through Microsoft Teams. Any other questions we had outside of meeting times were posted and answered through Teams chat. All team members participated in scheduled meetings and if they could not make it, promptly informed the team so the meeting could be rescheduled.

2.2.1 Meeting Times

In-person meeting: Monday at 3:00-3:30 pm

Online Teams meeting: Friday at 8:00 pm

2.3 Deliverables

- Project Selection Document
- Project Plan
- Software Requirements Specification
- System Design Document
- Prototype Presentation
- Weekly Activity Reports
- Project Website
- Final Report

2.4 Project Schedule

Project Name:	SP-5 Red Grocery List																	
Report Date:	4/27/2024																	
					Milestone #1				*Milestone #2*				*Milestone #3*				Final Report	
Phase	Tasks	Complete%	Current Status	Assigned To	01/29	02/05	02/12	02/19	02/26	03/04	03/11	03/18	03/25	04/01	04/08	04/15	4/22	04/27
Requirements	Establish project plan	100%	Completed	Aaron, Cole, Elizabeth	3													
	Define project requirements	100%	Completed	Aaron, Cole, Elizabeth		5	3											
	Finalize requirements	100%	Completed	Aaron, Cole, Elizabeth			3											
Design	Specify technology needed	100%	Completed	Cole, Elizabeth			8											
	User interaction flow chart design	100%	Completed	Aaron				4	4									
	UI/UX design	100%	Completed	Aaron				4	4									
	Database design	100%	Completed	Aaron, Cole, Elizabeth					5	10								
	Create offline prototype application	100%	Completed	Cole						5	5							
Development	Implementing basic layout of app	100%	Completed	Cole, Elizabeth							8							
	Connect application to database	100%	Completed	Cole							6	5						
	Add functionallity to application	100%	Completed	Cole							6	5	8					
	Test functionality of application	100%	Completed	Aaron, Cole, Elizabeth								3	3					
Final Report Package	Presentation preparation	100%	Completed	Aaron, Cole, Elizabeth								3	4					
	Create website	100%	Completed	Elizabeth									1	2				
	Completion and submission of final report	100%	Completed	Aaron, Cole, Elizabeth											4	4	3	3
			Total work hours	131	3	5	14	8	13	15	25	16	16	2	4	4	3	3

Legend	
Planned	
Delayed	
Number	Work: man hours

3.0 Requirements

3.1 Functional Requirements

3.1.1 Functional Requirement 1

The application allows users to create and modify lists.

3.1.1.1 Description

The application allows users to create and name multiple lists within the app. Whenever a user creates a list, they are registered as the owner of that list.

3.1.2 Functional Requirement 2

The application allows users to delete lists.

3.1.2.1 Description

Users should be able to manage their lists by deleting ones they no longer want. Furthermore, only users registered as owners should be able to delete those lists. Users invited to view and/or edit should not be able to delete the list.

3.1.3 Functional Requirement 3

The application allows users to share lists with other users.

3.1.3.1 Description

Users who are registered as owners of the lists should be able to invite selected users to view and edit their lists.

3.1.4 Functional Requirement 4

The application updates both users' views of the lists in real time.

3.1.4.1 Description

The application should be able to update the lists in real time. This ensures that any updates one user makes to a shared list are seen immediately by the other user.

3.1.5 Functional Requirement 5

The application is multiplatform.

3.1.5.1 Description

The application should be available and function equally on both the iOS and Android operating systems via Flutter.

3.1.6 Functional Requirement 6

The application has multiple views in the UI to access.

3.1.6.1 Description

The application should have multiple views, one for the user's available lists and another to view or edit a single list.

3.1.7 Functional Requirement 7

The application allows users to create and store accounts in Firebase.

3.1.7.1 Description

Users can create an account using an email and their own password. They can then use this account to share lists with other users. Once the account is created users will be able to set their own display name.

3.1.8 Functional Requirement 8

The application has a login system that utilizes created accounts.

3.1.8.1 Description

The application has a login interface to allow users to access and share lists. It should also allow users to request a password change if they forget their original one.

3.1.9 Functional Requirement 9

The application has a search function to search for specific items in grocery lists.

3.1.9.1 Description

The application allows users to search for keywords in their grocery lists.

3.2 Non-Functional Requirements

3.2.1 Security

Application has security features implemented to both secure the application and user data.

3.2.2 Encryption

The application has a form of encryption to ensure the confidentiality and integrity of data as it is sent between the application and the database.

3.2.3 Authentication

The application ensures that user accounts and their lists cannot be accessed without the appropriate login details. This guarantees that malicious users cannot tamper with or delete the lists of other users.

3.2.4 Capacity

Application can function seamlessly for multiple users using the databases concurrently.

3.2.5 Usability

Application is intuitive for users to operate and navigate.

3.2.6 UI

The UI has a pleasing look and simplistic feel that ensures the app is easy to use. The design should be able to work for several types of mobile devices.

3.2.7 Sorted Lists

The application allows users to sort lists in order of last updated. This ensures that lists more recently created or updated by the user will be at the top for easy access.

3.2.8 Other

The application has a feature to notify users when changes are made to lists that they created or have permission to edit. The notifications should be set to occur at reasonable intervals and not at every minor change to ensure the other user's device is not sent too many notifications.

4.0 Analysis

Overall, the project's goals were achieved with the intention of providing the best user experience for organized grocery shopping. To do this, we utilized multiple tools to design and develop our application. These tools being: Figma, Flutter, Firebase, GitHub, and Visual Studio Code.

4.1 Figma

To create the design for our UI of GrocAgree, we utilized Figma. Figma is a web-based design tool that can be used to collaborate with others to create designs in real-time. We utilized it to come up with the general layout and UI options in our application before development began.

4.2 Flutter

Our application is created with Flutter primarily with Android in mind but has potential to work on iOS as well. By using Flutter, we were able to test our application in real-time as changes were being made, this allowed us to immediately examine and test functionalities in the application without having to boot-up a device each time a small change was made.

4.3 Firebase

We decided to utilize Firebase due to its variety of tools and services that it provides its user. It offers reliable real-time management of user accounts, backend infrastructure, and allows for seamless integration with Flutter due to it also being developed by Google. With it, we were able to help simplify our application's development by utilizing its SDKs to implement database access and user authentication.

4.3.1 Firebase Firestore

With Firebase, we had two different databases as options: Firestore Database and Realtime Database. In the end, we decided on the use of Firestore, this decision was made after utilizing each database and deciding based its ease of use.

Firestore itself is a flexible NoSQL cloud-based database. It is structured using collections that store documents with fields. In our case, our collections were 'users' and 'lists'. The 'users' collection stores a document of each user that contains the fields of their email

and user ID. Another reason why Firestore was chosen is due to its scalability thanks to Google, it can easily be used for a small number of users or a large number.

4.3.2 Firebase Authentication

Firebase authentication allowed us to utilize an easy to integrate and secure form of authentication for accounts that utilize GrocAgree. Firebase Authentication gave us multiple options on how accounts are created and utilized. Firebase authentication also provides app developers to utilize its functionalities of password and email changing and SMS verification.

We decided to utilize the basic email and password sign-in method, but there are the options to use multi-factor authentication if we decided to.

4.4 GitHub

GitHub was used for collaboration between team members and to handle version control in the application's development. We created and utilized a shared GitHub account to access the code and to commit any changes. It allowed us to have the code be easily accessible and allowed for comments to be created to alert each team member of any large changes made.

4.5 Visual Studio Code

We utilized Visual Studio Code to code our application both due to the familiarity of it for each group member and due to its ability to code in various languages such as Dart which we utilized as our coding language for our Flutter Application. We utilized its ability to create Android emulations to troubleshoot and test our application. Visual Studio Code also has built-in Git integration that allows us to easily push and commit changes.

5.0 Design

5.1 User Interface Design

The user interface was designed to be user-friendly, visually appealing, and adaptable. Our design encompasses four primary sections: the login screen, the page displaying all user lists, another section for editing, viewing, and sharing lists with others, and finally a search feature for locating keywords within grocery lists.

We used Flutter to build the UI for the application. Using Flutter, we were able to design the app with helpful icons and pleasing colors. Our goal in designing the UI was to make the app simplistic looking so it doesn’t appear too crowded with features.

5.2 User Experience Design

When it came to designing the way users interact with the app, we wanted to make sure it had a comfortable and easy to use design. After users log in to the app, they are first met with the page where they can view all their lists. The lists are separated on the page by two headings called “My Lists” and “Shared Lists”. This ensures the user can easily find lists they have created and those they have been added to. The page was designed in such a way that users can scroll through each section independently. This means that even if the user has many lists created by them, it will not push the “Shared Lists” section down the page and out of the user’s view.

The main page also has buttons for deleting lists, signing out, creating lists, and searching for specific items or lists. The buttons each have icons or short labels showing what they do so users can easily figure out the app’s functionalities. The delete button puts the lists page into an “editing mode” where users can click on delete icons next to each list they wish to remove. They can delete as many lists as they want and then click a button to exit the “editing mode”.

The create list button opens a new page containing labeled text boxes so users know where to enter the list name and the items for the list. At the bottom of this page is a section showing all users the list is shared with. After this section there are two buttons, one which allows users to add users to the list, and one which publishes changes to the list. The button for adding a user to the list opens a pop-up window which prompts users to enter the email of the user they wish to add to their list. Once the update list button is pressed, the list will appear in the shared lists section of the other user.

The search button in the corner brings users to a page where they can enter a keyword at the top and find what they’re looking for quickly. This function will search for both list names and items on lists. For example, users can type in milk and all the lists containing milk will appear in the results. They can also simply search for the name of a list they wish to view.

5.3 Database Design

The design of collections in Firebase Firestore is structured to efficiently organize and manage user data. There are two distinct collections in the application’s database, “users” and “lists”. The “users” collection is meticulously crafted, housing generated documents for each user UID, complemented by essential fields such as “UID” and “email”. Meanwhile, the lists” collection is thoughtfully structured, containing generated documents for lists, with crucial attributes like “UID”, “listItems” array, “listName”, and an array for “sharedID”. This design ensures seamless access and manipulation of user and list data, facilitating smooth interactions within the Firestore environment.

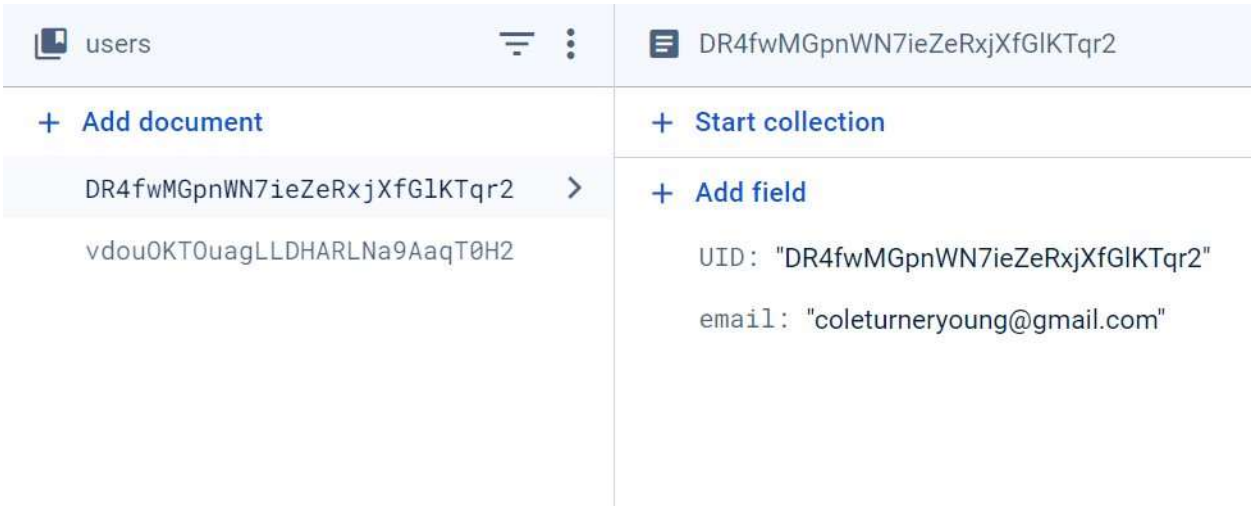


Fig. 5.3.1 Users Collection in the Database

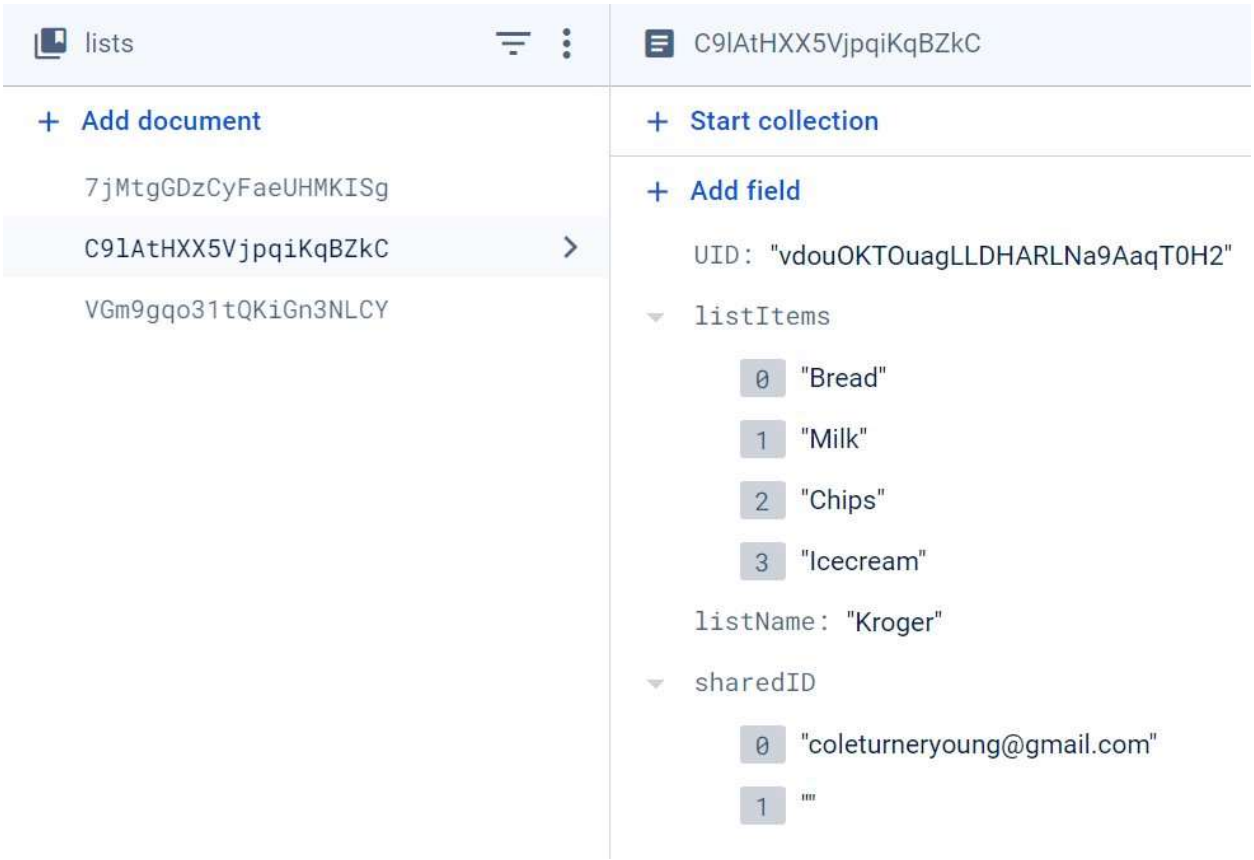
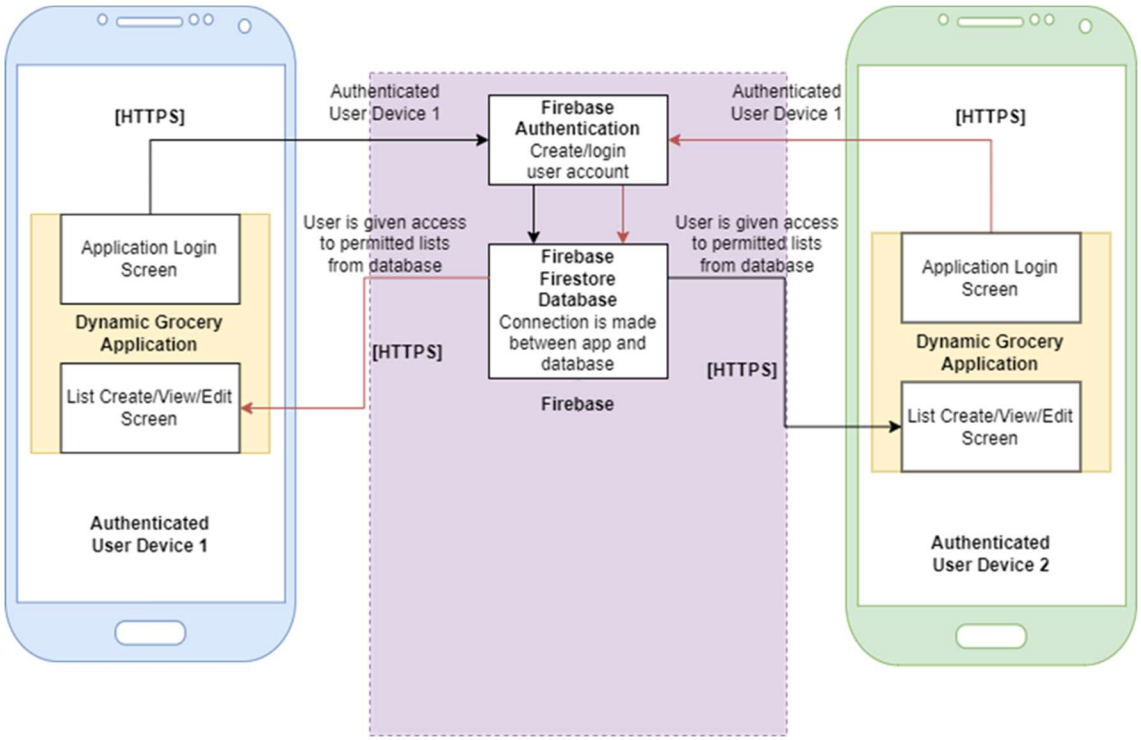


Fig. 5.3.2 Lists Collection in the Database

The database was also designed so that users can only delete lists they own. When a user deletes a list, the code checks to see if the list has been shared with them by another user. If it has, the list is simply removed from their view, but it is not deleted from the other user's available lists. If the list has not been shared with them, it can be deleted, and all data is removed from the database.

5.4 Software Architecture Design

Below is the process flow that we utilized to plan and develop our application GrocAgree. Users of the application must login or register with the application via Firebase Authentication. Upon logging in with a valid account, users are able to create/edit grocery lists that can then be shared with other authenticated users. All data being sent to and from GrocAgree is encrypted through the use of HTTPS with Firebase.



5.4.1 Process flow chart of application

5.5 Security

For our app's security, we created a login page where users must enter a valid email and password, which have been previously set up on the app's registration page, to access the app's data. Authentication is handled by Google Firebase's Authentication, which allows users to create an account using an existing email and set their own password. This ensures that only valid users can sign in and view the data.

Google Firebase also uses the HTTPS protocol to provide an extra layer of security for users' data going between the application and its servers.

6.0 Development

We took an object-oriented approach to ensure both flexibility and maintainability for our development process. We utilized various Dart files and functions to ensure that our code was modular and easily changeable all the while ensuring readability. This approach allowed us to easily adapt to any evolving requirements and additional functionalities without the risk of destabilizing the core functionality of our application. This methodology ensured that our application was flexible, well-structured, and with a solid foundation for any potential future changes or versions.

During development we first came up with the database design, and the types of data that our application will process and use. After deciding to use two collections called ‘lists’ and ‘users’, we connected our Flutter project to Firebase to set up authentication and the connection to the database. The registration and login, homepage, and widget tree for the application were implemented next.

The registration and login of GrocAgree is handled by an ‘Auth’ class. It connects to Firebase Authentication and utilizes methods to sign-in and create accounts. To create an account, it takes in an email address and a password. If a valid email is utilized, it will generate a user and a user ID. The user’s email and ID are then stored in the ‘users’ collection in Firestore. To sign-in, users enter their email and password that was created upon registration and the application’s widget tree is utilized.

The ‘widget_tree’ class navigates the user from the sign-in page to the home page if the user account data is authenticated. If a user signs out, to moves the user back to the login page.

The ‘home_page’ class handles the main screen that the user will be on when utilizing the application. This page includes buttons to sign-out, deleting existing grocery lists, creating grocery lists, search for keywords, and two scrolling lists that display: owned grocery lists and shared grocery lists.

The ‘search’ class allows users to search for items that exist on any list they own or have access to. Users are given the option to select a list to access or press the back button to return to the home page.

The ‘add_edit_list’ class allows users to create or edit grocery lists. It is accessed by either clicking on a grocery list or creating one on the home page. Users can enter the name of a list and list items. The user can also add and remove other users that can access the grocery list by entering or removing their email. Once a user is done creating/editing a list, there is a button for saving this list to Firebase under the user’s user ID or clicking on the back button to go back to the home page.

After testing the implemented authentication, we next added the main feature of the application, creating a grocery list that is saved to the database and linked with the list creator. Once we ensured that users could access and modify their lists, we added the ability for users to share lists with other users. We made it so only the list's creator can

delete it and implemented a search feature that looks for keywords in lists they have access to.

7.0 Testing

7.1 Test Plan

To test our application, we began by looking back at the requirements described in the SRS document. We then tested each function to make sure it worked as expected and did not contain any bugs that would cause a disruption in the app's functionality. Each function and the result of the test is described in the following test report.

7.2 Functional Requirements Test Report

The following report lists all functional requirements for the application as described previously in the SRS document. Under each requirement is a brief description of how the requirement works in the app followed by the result of the test performed on the function.

7.2.1 Functional Requirement 1

The application allows users to create and modify lists.

7.2.1.1 Description

The app has an add list button that allows users to name the list, provide its contents, and share it with other users. Users can also click on the names of existing lists and modify them by adding or removing items.

7.2.1.2 Test Result

When the add list button is clicked and the user enters the list name and items, the new list successfully appears under the users "Owned Lists" view. Lists can also be successfully modified by users.

7.2.2 Functional Requirement 2

The application allows users to delete lists.

7.2.2.1 Description

Users can delete lists fully if they own the list. If a user is looking at a list that was shared with them, the user can remove the list from their end, which will not delete the list entirely, but it will remove the list from their shared list's view.

7.2.2.2 Test Result

When the delete button is pressed on an owned list, that list and all its contents are successfully removed. When the delete button is pressed on a shared list, the user is successfully removed from the list of users it is shared with.

7.2.3 Functional Requirement 3

The application allows users to share lists with other users.

7.2.3.1 Description

Users who are authenticated through the app's database can share lists with other authenticated users via the user's email. This is done through typing in the email of the user they want to share the list with.

7.2.3.2 Test Result

When a user enters the email address of another user in the "Shared With" field, the list successfully appears under the "Shared Lists" view of that other user.

7.2.4 Functional Requirement 4

The application updates both users' views of the lists in real time.

7.2.4.1 Description

The app updates information entered by users in real time. If a list is shared with another user, that list will appear on their shared lists in real time. Also, any updates to a shared list can be viewed by both users as soon as they are made.

7.2.4.2 Test Result

When a user modifies a list that is shared with another user, any changes made are successfully seen by both users instantly.

7.2.5 Functional Requirement 5

The application is multiplatform.

7.2.5.1 Description

The application should be available and function equally on both the iOS and Android operating systems via Flutter.

7.2.5.2 Test Result

We were not able to develop and test the application on iOS due to limitations described in section 5.0 of this document.

7.2.6 Functional Requirement 6

The application has multiple views in the UI to access.

7.2.6.1 Description

The main view of the application shows both the lists created by the user and lists that have been shared with the current user. When clicking the button to add a list, the user is brought to another view where they can enter the list name, add items, and share the list with others. When an existing list is selected, the user is brought to a view where they can edit the list name, add or remove items on the list, and add or remove users the list is shared with. Finally, when the user clicks on the search icon, another view will appear that displays lists that match the user's search criteria.

7.2.6.2 Test Result

All views as described were successfully added to the application and work as intended.

7.2.7 Functional Requirement 7

The application allows users to create and store accounts in Firebase.

7.2.7.1 Description

When the app is first opened the user can click the register button to sign up for the app. The user can then enter an email and a password and click the register button underneath. The information they entered will be added to the list of registered users.

7.2.7.2 Test Result

When a user registers with the app for the first time, the login information they provide is successfully added to Firebase Authentication's list of users.

7.2.8 Functional Requirement 8

The application has a login system that utilizes existing accounts.

7.2.8.1 Description

Upon launching the app, a user is prompted to login to an existing account via username and password. If they do not have an existing account, they can create one with a username and password.

7.2.8.2 Test Result

Returning users can successfully log in and view their lists after entering the login information they provided when first registering for the app. If the information they entered

is not correct, a message will be displayed informing the user that it is not correct, and they can re-enter the information.

7.2.9 Functional Requirement 9

The application has a search function to search for specific items in grocery lists.

7.2.9.1 Description

A search icon can be clicked that allows users to enter either the name of a list or an item on a list that allows users to quickly find specific items or lists.

7.2.9.2 Test Result

Users can click on the search icon and enter an item or list they want to find. The available lists are searched and are filtered successfully based on the user's search criteria.

7.3 Nonfunctional Requirements Test Report

7.3.1 Sorted Lists

The application allows users to sort lists by date.

7.3.1.1 Description

The application has a button that allows users to sort lists in order of last updated. This ensures that lists more recently created or updated by the user will be at the top for easy access.

7.3.1.2 Test Result

Not implemented in this version.

7.3.2 Notifications

The application allows users to receive notifications about lists.

7.3.2.1 Description

The application has a feature to notify users when changes are made to lists that they created or have permission to edit. The notifications should be set to occur at reasonable intervals and not at every minor change to ensure the other user's device is not sent too many notifications.

7.3.2.2 Test Result

Not implemented in this version.

8.0 Version Control

In the development of our application, we used GitHub for project management and version control of our Dart code. It allowed us to work individually on individual branches without the risk of messing up the stability of our main branch. It also allowed us to do commits and pull requests to ensure that, as a team, we maintained quality, readable code to ensure a productive and collaborative group effort.

Our application was created and maintained through multiple Dart files that handle its connections to Firebase and the various pages of the application.

9.0 Future Changes

Moving forward, we are excited to explore a variety of enhancements aiming to level up the capabilities of our grocery list application. We are determined to expand on the quality and user experience of the app. We plan on working on two areas of focus: refining the application design and integrating real-time notifications. The notifications will keep users informed about interactions made with their lists which will enhance user experience and functionality of the app. By concentrating on future improvements, we aim to provide the best engaging experience for our users.

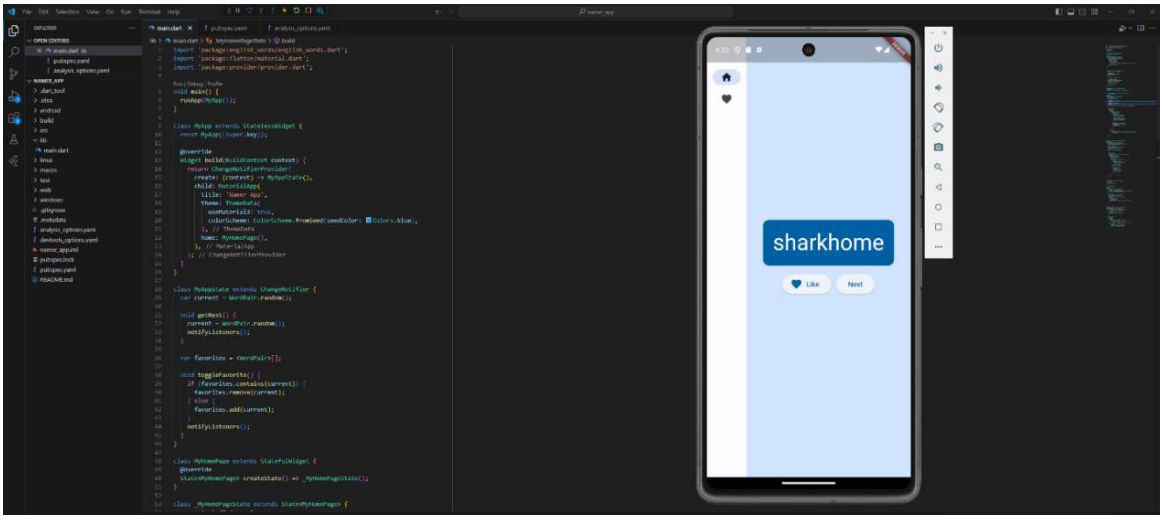
One of the main features we would like to add in the future is the ability to sort lists by data. We would like to allow users to sort lists by most recently created or updated so they can quickly access the most relevant grocery lists.

Another feature we want to add is to allow the app to send notifications. Users could enable notifications for the app to receive updates if any changes are made to shared lists or if they have been added to a new list. This would ensure users are up to date on the latest changes even if they are not on the app.

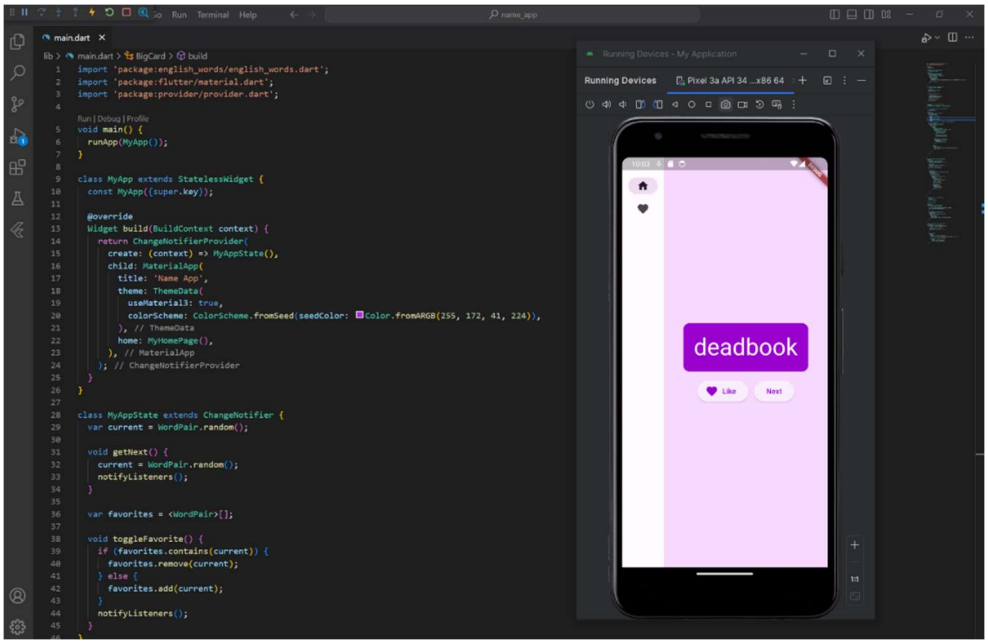
10.0 Conclusion

In summary, our grocery list application was made to represent a simple and efficient way of grocery management for our users. Utilizing our technologies including VS Code, Flutter, Dart, Google Firebase, GitHub, and Figma, we were able to develop an application that stands out for its ease of use and similarity to traditional written grocery lists. Not only is the application a great tool for managing groceries, but it also engages users to interact/collaborate with others who are also managing groceries. Our application highlights great use of features through real-time integration of functionalities that cater to user needs of creating, modifying, and sharing grocery lists.

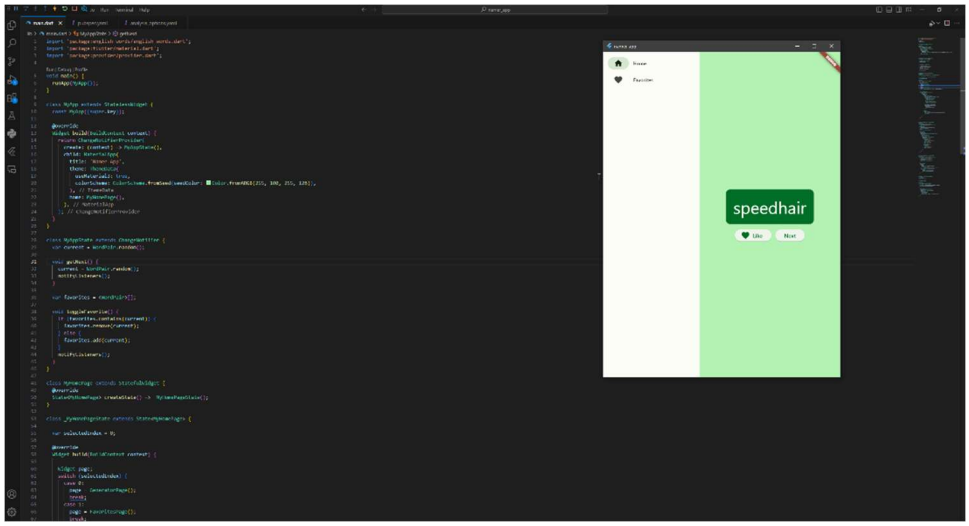
Appendix



Cole Young completed Flutter tutorial screenshot.



Elizabeth Krummert completed Flutter tutorial screenshot.



Aaron Figueroa completed Flutter tutorial screenshot.