

Final Report:

Course CS: 4850, Spring 2024

Semester: Spring 2024

Professor: Sharon Perry

Date: 4/14/24

Team: SP-5-Orange- Dynamic Grocery List

Website Link: https://sp5orange.github.io/Senior_Project_Website/

GitHub Link: <https://github.com/sp5orange/FlutterProject>

Number Of Lines of Code Est: 3,000

Number of Components: 40-50



Lee Martin



Richard Thomas



Jesse Schulteis



Yesahk Ergano

--	--

1.0 Introduction.....	5
1.1 Overview.....	5
1.2 Project goal.....	6
1.3 Definitions and Acronyms.....	7
1.4 Assumptions	8
2.0 Design Constraints	9
2.1 Environment	9
2.2 User Characteristics	10
2.3 System	11
3.0 Functional Requirements	12
3.1 Authentication and User Management.....	12
3.2 Grocery List Management.....	13
3.3 Item Management within Grocery Lists.....	13
3.4 Real-time Data Synchronization	13
4.0 Non-Functional Requirements	14
4.1 Security	14
4.1.1 Authentication Security	14
4.2 Capacity	14
4.2.1 Scalability	14
4.2.2 Database Capacity	14

4.3 Usability.....	15
4.3.1 User Interface Design	15
4.3.2 Accessibility	15
4.4 Other	15
4.4.1 Performance.....	15
4.4.2 Reliability.....	16
5.0 External Interface Requirements	16
5.1 User Interface Requirements.....	16
5.1.1 Intuitive Interface Design	16
5.1.2 Responsive Layout.....	16
5.2 Hardware Interface Requirements	16
5.2.1 Device Compatibility	17
5.2.2 Touchscreen Interaction.....	17
5.3 Software Interface Requirements.....	17
5.3.1 Integration with Firebase.....	17
5.4 Communication Interface Requirements	17
5.4.1 Network Connectivity	17
6. Development.....	18
6.1 Document Outline.....	22
6.3 Document Description	23
7. System Overview	24
8. Design Considerations	30
8.1 Assumptions and Dependencies	30
8.2 General Constraints.....	31
8.3 Goals and Guidelines.....	32
8.4 Development Methods.....	32
9. Architectural Strategies.....	33
10. System Architecture	34
10.1 Subsystem Architecture	35

10.2 Version Control	36
11. Analysis.....	37
12. Detailed System Design	38
12.1 Classification.....	38
12.2 Definition	38
12.3 Responsibilities	39
12.4 Constraints.....	39
12.5 Composition.....	39
12.6 Uses/Interactions.....	39
12.7 Resources	39
12.8 Processing.....	39
12.9 Interface/Exports.....	40
13. Testing plan.....	40
1. Introduction	40
2. Test Objectives	40
3. Test Environment	41
4. Test Scenarios	41
1. User Registration and Login	41
2. Creating a Grocery List.....	41
3. Managing Grocery Lists.....	41
4. Sync and Cloud Storage	41
5. Performance and Load Testing.....	42
6. Usability and Accessibility	42
7. Security	42
5. Test Execution	42
14. Test Report	42
14.1 User Registration and Login	42
14.2 Creating a Grocery List.....	42
14.3 Managing Grocery Lists.....	43

14.4 Sync and Cloud Storage	43
14.5 Performance and Load Testing	43
14.6 Usability and Accessibility	43
14.7 Security	43
15. Conclusion	44
15.1 Key Achievements	44
15.2 Lessons Learned	44
16. Appendix	45
16.1 Training	45
17. Glossary	48
18. Bibliography	48

Table of Contents

1.0 Introduction

Welcome to the Software Requirements Specification (SRS) document for the development of an innovative dynamic grocery list mobile application. This document serves as a comprehensive guide outlining the essential specifications and functionalities required for the creation of this application. The dynamic grocery list app is envisioned to provide users with a cutting-edge solution for efficiently managing their grocery lists, offering a seamless and responsive user interface.

1.1 Overview

Our dynamic grocery list mobile application aims to revolutionize the way users manage their shopping needs by providing a comprehensive and intuitive solution accessible via mobile

devices. Developed using the Dart programming language and Flutter framework, the application will offer a seamless and responsive user interface tailored for both iOS and Android platforms.

At its core, the application will empower users to create, customize, and organize multiple grocery lists effortlessly. Leveraging Firebase for real-time data synchronization, any changes made to grocery lists will be instantly updated across all devices, ensuring consistency and collaboration among users.

Key features of the application include user authentication, list creation and management, item addition and removal, and real-time synchronization. Through integration with Firebase Firestore, the application will provide a reliable and scalable backend infrastructure to support data storage and synchronization.

Overall, the project seeks to deliver a dynamic grocery list mobile application that enhances user convenience, efficiency, and collaboration in managing their shopping lists, ultimately improving the overall shopping experience for users.

1.2 Project goal

The overarching goal of the dynamic grocery list mobile application project is to provide users with a streamlined and user-friendly solution for managing their grocery lists effectively. At its core, the application aims to simplify the process of creating, updating, and organizing grocery lists, enhancing the overall shopping experience for users.

Central to the project is the utilization of Flutter for frontend development and Firebase for real-time data synchronization. By leveraging these powerful technologies, the application endeavors to deliver a seamless and responsive user interface that caters to the diverse needs of modern consumers.

One of the primary objectives of the project is to empower users with the ability to create and customize multiple grocery lists effortlessly. Whether planning for a weekly shopping trip, organizing ingredients for a specific recipe, or maintaining a running inventory of household essentials, the application will provide intuitive tools for managing various types of lists.

Real-time data synchronization through Firebase is a pivotal aspect of the project, ensuring that any changes made to grocery lists are instantly propagated across all devices. This feature not only enhances user convenience but also facilitates collaboration among family members or housemates who may share shopping responsibilities.

Another key goal of the project is to prioritize user experience by implementing a responsive and visually appealing interface. By adhering to platform-specific design guidelines and leveraging Flutter's capabilities for building native-like applications, the dynamic grocery list app aims to deliver an immersive and delightful experience for users on both iOS and Android platforms.

Overall, the project goal is to create a dynamic grocery list mobile application that not only simplifies the task of shopping but also enhances the way users plan, organize, and manage their grocery needs. Through innovation, usability, and seamless integration, the application aims to become an indispensable tool for modern consumers seeking convenience and efficiency in their shopping endeavors.

1.3 Definitions and Acronyms

- **Dynamic Grocery List Mobile Application:** Refers to the mobile application developed using Flutter and Firebase, allowing users to create, customize, and manage grocery lists in real-time.
- **Flutter:** A UI toolkit developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase.
- **Firebase:** A comprehensive mobile and web application development platform provided by Google, offering various services such as authentication, real-time database, cloud storage, and hosting.
- **Dart:** The programming language used for developing the dynamic grocery list mobile application, known for its flexibility, performance, and ease of learning.
- **SRS:** Software Requirements Specification, a document outlining the functional and non-functional requirements for a software project.
- **SDK:** Software Development Kit, a set of tools and libraries used for developing applications for a specific platform or programming language.

- **CRUD:** Create, Read, Update, Delete, referring to the basic operations performed on data within the application.
- **UI:** User Interface, the visual elements and design of the application that users interact with.
- **UX:** User Experience, the overall experience and satisfaction users derive from using the application.
- **API:** Application Programming Interface, a set of protocols and tools for building software applications.
- **Firestore:** Firebase's cloud-native, scalable database for storing and syncing data in real-time.

1.4 Assumptions

- **Stable Internet Connectivity:** It is assumed that users will have consistent access to a stable internet connection to enable real-time data synchronization with Firebase. This assumption ensures that updates made to grocery lists are promptly reflected across all devices, enhancing user experience and collaboration.
- **User Adoption:** The success of the application relies on users' willingness to adopt and engage with the platform. It is assumed that users will find value in the features offered by the dynamic grocery list app, leading to widespread adoption and usage.
- **Flutter and Dart Support:** The project assumes ongoing support and updates for Flutter and Dart programming language. This assumption ensures compatibility with future platform updates and maintains the application's performance and functionality.
- **Third-Party Integrations:** Dependencies on third-party libraries or plugins are assumed to function as expected. This includes integrations for additional functionalities such as analytics, authentication, and user engagement tools.
- **User Experience Expectations:** It is assumed that users will have certain expectations regarding the user experience, including intuitive navigation, responsive design, and visually appealing interfaces. Meeting these expectations is crucial for user satisfaction and retention.
- **Device Compatibility:** The project assumes that the dynamic grocery list mobile application will be compatible with a wide range of mobile devices running on both iOS and Android platforms. This assumption ensures that the application can cater to a diverse user base and maximize its reach and accessibility.

- **Data Security:** It is assumed that Firebase provides robust security measures to safeguard user data stored within the application. This includes measures such as data encryption, user authentication, and access control to protect sensitive information from unauthorized access or breaches. Users trust that their personal information and grocery lists are securely managed and protected by the application.

2.0 Design Constraints

The design of the dynamic grocery list mobile app is subject to several constraints to ensure its functionality, usability, and performance. Firstly, the app must adhere to platform-specific design guidelines for iOS and Android platforms, ensuring consistency with the respective user interface conventions. Additionally, the design elements must be responsive to accommodate various screen sizes and resolutions, maintaining usability across a diverse range of mobile devices. Optimization for performance is crucial to minimize load times and resource consumption, enhancing the overall user experience. Furthermore, real-time data synchronization with Firebase introduces constraints related to network latency and reliability, necessitating robust error handling mechanisms and synchronization strategies to ensure seamless data updates.

2.1 Environment

The dynamic grocery list mobile application operates within a multifaceted environment, encompassing various hardware and software components, as well as user interaction scenarios.

Hardware Platform: The application targets both iOS and Android platforms, accommodating a wide range of mobile devices such as smartphones and tablets. It must adapt to different screen sizes, resolutions, and hardware capabilities to deliver a consistent and optimal user experience across various devices.

Operating Systems: Compatibility with multiple versions of iOS and Android is essential to ensure broad accessibility and reach among users. The application should seamlessly integrate with the latest features and functionalities offered by the respective operating systems to enhance usability and performance.

Geographical Locations: The application is designed to be accessible globally, enabling users from different regions and countries to utilize its features for managing their grocery lists. It must accommodate diverse cultural and linguistic preferences, including support for multiple languages and regional settings.

Network Connectivity: The application's functionality relies on stable internet connectivity for real-time data synchronization with Firebase. It must gracefully handle scenarios of limited or intermittent network access, providing offline capabilities where feasible to ensure uninterrupted usage and data integrity.

User Environment: The environment also includes the context in which users interact with the application, such as their physical surroundings, preferences, and behaviors. Understanding user contexts and needs is essential for designing intuitive and efficient user interfaces that align with users' expectations and shopping habits.

Overall, the environment in which the dynamic grocery list mobile application operates is dynamic and diverse, requiring careful consideration of various factors to ensure optimal performance, usability, and user satisfaction.

2.2 User Characteristics

The mobile app for managing grocery lists is designed to accommodate a wide range of users, encompassing various demographics, levels of technical expertise, and shopping preferences.

Technical Proficiency: Users of the application may range from tech-savvy individuals familiar with mobile applications to those less experienced with digital technology. Design considerations must accommodate varying levels of technical proficiency, offering intuitive interfaces and guidance to facilitate seamless interaction with the app.

Shopping Habits: Users exhibit a range of behaviors and preferences when it comes to grocery shopping. Some users may prefer to plan meticulously, creating detailed lists for specific recipes

or dietary requirements, while others may opt for more spontaneous shopping trips. The application must accommodate different shopping styles, offering flexibility and customization options to meet diverse needs.

Accessibility Requirements: Consideration must be given to users with accessibility needs, such as those with visual or motor impairments. The application should adhere to accessibility standards and guidelines to ensure inclusivity and usability for all users, regardless of their abilities.

In summary, the dynamic grocery list mobile application caters to a diverse user base with varying characteristics and needs. By understanding and accommodating these characteristics, the application can provide a personalized and engaging experience that meets the requirements and expectations of its users.

2.3 System

The dynamic grocery list mobile application comprises several interconnected components that work together to deliver a seamless and feature-rich user experience.

Frontend: Developed using the Flutter framework, the frontend of the application is responsible for rendering the user interface and handling user interactions. It incorporates platform-specific design elements and follows UI/UX best practices to ensure a visually appealing and intuitive interface for users. The frontend interacts with the backend services to fetch and display grocery list data, as well as facilitate user actions such as list creation, item addition, and list sharing.

Backend: Leveraging Firebase, the backend of the application provides a scalable and reliable infrastructure for storing and synchronizing data in real-time. Firebase Firestore serves as the database system, storing grocery list information such as list names, item details, and user preferences. Firebase Authentication is used for user authentication and authorization, ensuring secure access to personal data and preventing unauthorized access. The backend services handle data synchronization between the client-side application and the cloud, ensuring that updates made to grocery lists are propagated instantly across all devices.

Database: Firebase Fire store serves as the primary database system for storing and querying grocery list data. It provides a scalable, cloud-native solution for storing structured data and enables real-time synchronization of updates across devices. The database schema includes collections for storing grocery lists, documents for individual lists containing item details, and subcollections for user-specific data such as preferences or shared lists.

Data Synchronization: Real-time data synchronization is a core feature of the system, facilitated by Firebase Fire store. Changes made to grocery lists by one user are immediately synchronized and reflected on all devices connected to the application. This ensures consistency and collaboration among users, enabling seamless sharing and updating of grocery lists across multiple accounts.

In summary, the dynamic grocery list mobile application's system architecture encompasses frontend and backend components that work together to provide a responsive, scalable, and secure platform for managing grocery lists effectively. Through integration with Firebase services, the system delivers real-time data synchronization and user authentication capabilities, enhancing the overall usability and functionality of the application.

3.0 Functional Requirements

3.1 Authentication and User Management

Design Requirement: The application must provide a user-friendly authentication interface for users to log in and sign up securely.

Graphic Requirement: The authentication interface should feature clear and intuitive design elements, such as input fields for email and password, as well as options for social login.

Operating System Requirement: The authentication process should be consistent with platform-specific design guidelines for both iOS and Android, ensuring a cohesive user experience.

Constraint: Authentication must be implemented using Firebase Authentication to ensure secure user authentication and authorization.

3.2 Grocery List Management

Design Requirement: The application must allow users to create, edit, and delete multiple grocery lists.

Graphic Requirement: Grocery lists should be visually distinct and easily recognizable, with options to customize list names and colors.

Constraint: Each grocery list should have a unique identifier and be associated with the user's account to facilitate data synchronization and access control.

3.3 Item Management within Grocery Lists

Design Requirement: Users should be able to add, remove, and update items within each grocery list.

Graphic Requirement: Item entries should include fields for item name, quantity, unit, and optional notes for additional details.

Operating System Requirement: Item management actions should be easily accessible within the grocery list interface, allowing users to perform operations with minimal effort.

Constraint: Item data should be stored in a structured format within the Firebase Firestore database, enabling efficient querying and synchronization across devices.

3.4 Real-time Data Synchronization

Design Requirement: Changes made to grocery lists or items should be instantly synchronized across all devices in real-time.

Graphic Requirement: Synchronization status indicators should be displayed to users, indicating whether data is up-to-date or pending synchronization.

Operating System Requirement: Synchronization mechanisms should be implemented using Firebase Fire store's real-time database capabilities, ensuring seamless data updates across devices.

Constraint: Users must have stable internet connectivity to facilitate real-time data synchronization with Firebase, with appropriate error handling mechanisms in place for offline scenarios.

4.0 Non-Functional Requirements

4.1 Security

Data in-transit – Fire store uses HTTPS and TLS to securely deliver the data to the database.

Data at rest – Fire store uses AES 256-bit encryption with rotating keys that rotate to ensure computationally secure data.

4.1.1 Authentication Security

Requirement: User authentication must be secure and robust, utilizing industry-standard encryption protocols to protect user credentials during login and registration processes.

Constraint: Firebase Authentication will be used to handle user authentication, ensuring secure access to user accounts and sensitive data.

4.2 Capacity

4.2.1 Scalability

Requirement: The application must be able to handle many concurrent users and scale dynamically to accommodate increased user demand over time.

Constraint: Firebase's scalable infrastructure ensures that the application can handle increased traffic and data volume without compromising performance or reliability.

4.2.2 Database Capacity

Requirement: The database system must have sufficient capacity to store and manage user-generated data, including grocery lists, items, and user preferences.

Constraint: Firebase Fire store offers scalable database storage, allowing the application to store large amounts of structured data efficiently and cost-effectively.

4.3 Usability

4.3.1 User Interface Design

Requirement: The user interface must be intuitive, with clear navigation paths, easily identifiable actions, and consistent design patterns across different screens.

Constraint: Design elements and navigation patterns should adhere to platform-specific guidelines for iOS and Android, providing a familiar experience for users on each platform.

4.3.2 Accessibility

Requirement: The application must be accessible to users with disabilities, incorporating features such as screen reader support, keyboard navigation, and high contrast modes.

Constraint: Accessibility features should comply with relevant accessibility standards and guidelines, ensuring inclusivity and usability for all users.

4.4 Other

4.4.1 Performance

Requirement: The application must be responsive and performant, with fast loading times, smooth animations, and minimal latency for user interactions.

Constraint: Optimization techniques such as code minification, image compression, and caching should be employed to optimize performance across different devices and network conditions.

4.4.2 Reliability

Requirement: The application must be always reliable and available for use, with minimal downtime or service interruptions.

Constraint: Firebase's robust infrastructure and automatic failover mechanisms ensure high availability and reliability, minimizing the risk of service disruptions or outages.

5.0 External Interface Requirements

5.1 User Interface Requirements

5.1.1 Intuitive Interface Design

Requirement: The user interface must be intuitive and user-friendly, featuring clear navigation paths, easily identifiable actions, and consistent design elements.

Constraint: Design elements should adhere to platform-specific guidelines for iOS and Android, ensuring a cohesive user experience across different devices and operating systems.

5.1.2 Responsive Layout

Requirement: The user interface must be responsive, adapting seamlessly to various screen sizes and orientations to provide an optimal viewing and interaction experience.

Constraint: Design layouts should be flexible and adaptive, utilizing responsive design techniques such as fluid grids and media queries to accommodate different device resolutions and aspect ratios.

5.2 Hardware Interface Requirements

5.2.1 Device Compatibility

Requirement: The application must be compatible with a wide range of mobile devices, including smartphones and tablets running on iOS and Android platforms.

Constraint: Hardware interfaces should be implemented to ensure compatibility with device-specific features and capabilities, such as camera access for barcode scanning or location services for store locator functionality.

5.2.2 Touchscreen Interaction

Requirement: The application must support touchscreen interaction, allowing users to navigate, scroll, and interact with content using touch gestures such as tapping, swiping, and pinching.

Constraint: Touchscreen interactions should be implemented using platform-specific touch events and gestures to provide a seamless and intuitive user experience across different devices and operating systems.

5.3 Software Interface Requirements

5.3.1 Integration with Firebase

Requirement: The application must integrate with Firebase services for authentication, data storage, and real-time synchronization.

Constraint: Software interfaces should be implemented to interact with Firebase Authentication, Fire store database, and Cloud Messaging services, enabling seamless integration and communication with Firebase backend infrastructure.

5.4 Communication Interface Requirements

5.4.1 Network Connectivity

Requirement: The application must support network connectivity for real-time data synchronization, remote authentication, and communication with external services.

Constraint: Communication interfaces should be implemented using secure protocols such as HTTPS for data transmission and Firebase Cloud Messaging for push notifications, ensuring reliable and encrypted communication over the network.

6. Development

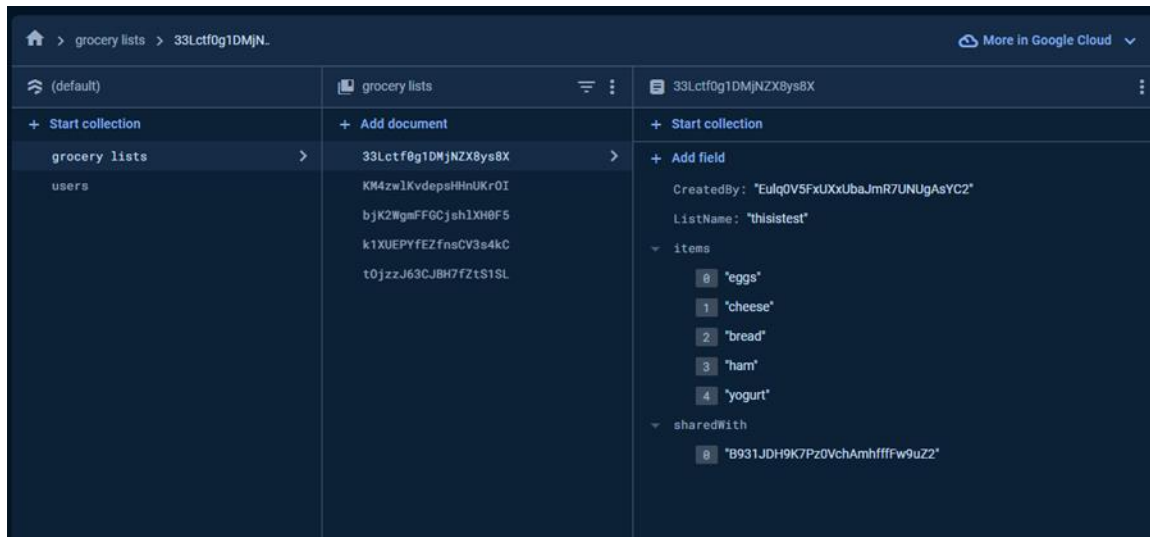
In the beginning we reviewed all the project scope and started to do some brainstorming and gather ideas. We had to evaluate every different type of language and framework based on everyone's current and experience and what would be the best to benefit us for our future goals. As a team we generated vague requirements and specifications and did not want to have any scope creep on this project. The requirements and all the documentation were handled in conjunction with Yesahk and Lee. We decided that we would use Flutter and Dart early on and all came together using the entire google ecosystem which included Firebase/Fire store.

In development during the requirements and specification we knew at a minimum that we needed to implement CRUD operations and that was a given. The faster that we could get the database reading, writing, and deleting the more wholesome of a presentation and demo we could prepare for the prototype presentation.

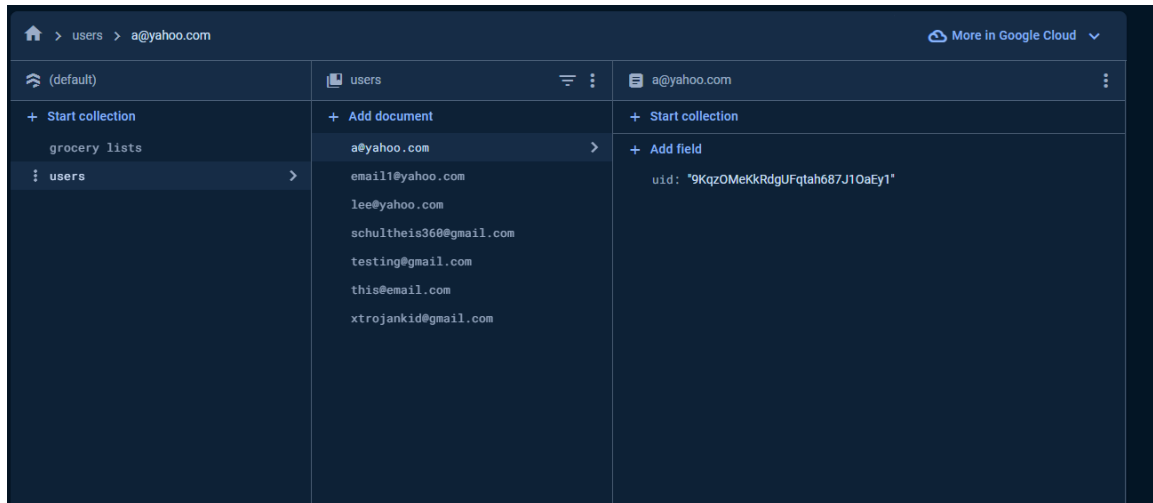
We created a GAANT chart and got a rough estimate of what we were aiming to do for each day and how many hours that we would dedicate to each item. We agreed that it would be much simpler to move in sprints and came to an agreement that UI needed to be done first. The project lead (Lee Martin), decided to give a rough UI to follow in Figma, then shortly started and learned the flutter framework to quickly develop the UI for the project. Once the visuals were developed it was then another sprint to get the CRUD operations functionals with the bare minimum visuals that was created that was done by Richard, and Jesse.

From there we created multiple different pages, consisting of a create account, login, view list, update list, and add user (used for sharing). We created the layout for the database, which consisted of multiple collections starting with grocery list and users. The users have an account email and a UID that would be defined to reference for authentication, and to add everyone under the specified list that they have created – this is also used when you want to share an item to another user. It will then pull the UID and compare it uniquely identify it to add it to the specified users list (under shared).

Users and Grocery List NoSQL Database (Firestore):



Users and Grocery List NoSQL Database (Firestore):



From there we built it out to be able to source this data inside of the create account and login pages using emailControllers and passwordControllers that we can source from the build context (which the UI visuals). It should also be noted that we need to have an auth instance from Firebase to be able to secure this data through HTTPS and TLS to authenticate.

```
final TextEditingController _emailController = TextEditingController();
final TextEditingController _passwordController = TextEditingController();

Future<void> _createUserWithEmailAndPassword(BuildContext context) async {
  try {
    final FirebaseAuth auth = FirebaseAuth.instance;
    UserCredential userCredential = await auth.createUserWithEmailAndPassword(
      email: _emailController.text.trim(),
      password: _passwordController.text,
    );

    if (userCredential.user != null) {
      if (userCredential.user != null) {
        final String email = userCredential.user!.email!;
        final String uid = userCredential.user!.uid;

        await FirebaseFirestore.instance.collection('users').doc(email).set({
          'uid': uid,
        });

        Navigator.of(context).pushReplacement(
          MaterialPageRoute(
            builder: (context) => const ViewListsPage(),
          ), // MaterialPageRoute
        );
      }
    }
  }
}
```

From there we can check if the user Credential is not null if it is then we can go ahead and proceed with authentication and try to match the ID. If we can get to the bottom of that method that means we can go ahead and push the credentials and state to the View Lists Page. This is the exact way that we also authenticate for the login page.

For the Create List page we have a listNameController and a descriptionController that will be used inside of the createGroceryList method that will pull from the existing build context. It will allow us to create a name and description and source it from the context. We will also split the description of the list to be separated by commas. This will all be stored with cards inside of a list view for the view page to be able to scroll and access multiple lists. You are also now able to edit the list, swipe to delete, and as you check the item to delete it will automatically bump it to the bottom of the list of items. There is a sign out button that appears on the main screen of the view lists page and it will use the auth instance to terminate the session for the user.

```
class CreateList extends StatelessWidget {
  CreateList({super.key});

  final TextEditingController _listNameController = TextEditingController();
  final TextEditingController _descriptionController = TextEditingController();

  Future<void> createGroceryList(String listName, String description) async {
    final uid = FirebaseAuth.instance.currentUser?.uid;
    final items = description
      .split(',')
      .map((e) => e.trim())
      .toList();

    await FirebaseFirestore.instance.collection('grocery lists').add({
      'CreatedBy': uid,
      'ListName': listName,
      'items': items,
      'sha': Type: List<dynamic>
    });
  }
}
```

Inside of the Fire store rules we have limited querying and modified the access to further validate the user to a UID to increase security to also ensure that the user that is requesting information has already been authenticated to reduce the chance of reverse engineering, or an attack gather data.

```

rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    //this is what it was before
    //allow read, write: if request.time < timestamp.date(2024, 4, 10);

    //allow read and write if false
    //allow read, write: if false;

    // Existing rules for user-specific access control
    match /users/{userID}/{document=**} {
      allow read, write: if request.auth.uid == userID;
    }

    // Additional rule to limit the size of queries to prevent large data retrievals
    match /users/{userID}/notes/{document=**} {
      // Limit the number of documents retrieved in a query to prevent abuse
      allow list: if request.query.limit <= 50;
    }

  }
}

```

Inside of the application we must add another item which will automatically pull you to a create new list and will follow the format above. You will also be able to delete the list, update the list, or share the list which will reference the UID from the specified email that is provided to share that grocery list – that field alone will have a separate collection called shared with that will reference the UID that is being added.

In the final stages we have added an easier to read UI, but not much has changed since we have been operating and have been completing our sprints to complete each functionality. We have added cleaner photos and an easier interface for the application. It should be noted that this app works on both Android and Apple after importing the right API, and SDKs for the “build-gradle” and “google.json” that holds the API key to talk to Firebase.

Some of the early challenges came down to design and how we would get everything linked “seamlessly”. Through countless hours of researching, we decided to keep the google ecosystem and surprisingly linking Firebase was not too difficult if Google provides friendly-easy to read documentation for anything that you want to implement using their API – they also include YouTube videos.

6.1 Document Outline

- Introduction
- System Overview
- Design Considerations
 - Assumptions and Dependencies

- General Constraints
 - Goals and Guidelines
 - Development Methods
- Architectural Strategies
- System Architecture
- Analysis
- Detailed System Design
- Glossary
- Bibliography

6.3 Document Description

- Introduction – This is going to include the introduction to this document and discuss the overall goal of this document, in accordance with the addition of other documents to support the design and development of this piece of software: Dynamic Grocery List.
- System Overview – Discuss the high-level overview of the system, the functionality, and what this would look like/mean to users. This will aim to provide a more comprehensive overview of how the software will perform/achieve.
- Design Considerations – Discuss the overall design choices following these subtopics:
 - - Assumptions and Dependencies – Assumptions made by our group but also an understanding of what this project will depend on.
 - General Constraints – This will describe any hard constraints that will keep this project/software design within a certain boundary, whether that be software, technology, or student resources.
 - Goals and Guidelines – This aims to announce the goals of this software in regard to design and the guidelines that we must follow in order to develop successful software.
 - Development Methods – This will discuss the type of software development process that our group aims to use during the development process.
- Architectural Strategies – This will discuss whether our group has chosen to use/might depend on any microservices that will affect the non-functionality of this software.
- System Architecture – This will describe how different parts of the system will interact/interface with each other. It will discuss each service and how that will be related to providing a more holistic product.
- Policies and Tactics – This will describe/approach the policies that we have used or considered to affect the non-functional requirements such as scalability/reliability/ and security.

- Detailed System Design – This will go into low-level detail regarding the system design and aim of our software.
- Glossary – Our section that will define any technical terms used in this document for others to help understand.
- Bibliography – This will list all references that we have used to complete this entire software design document, or anything that aims in the finish of this document.
- Purpose of this document – The purpose of this document is to provide a complete design of the dynamic grocery list piece of software. This will guide developers and users/stakeholders for future relevance.
- The scope of this document – The scope of this document will go over the design and the architectural design of our dynamic grocery list application – that will extend from our requirements design in terms of functional and non-functional requirements. It will help give an understanding of our entire software development life cycle.
- Intended audience – The intended audience for this document will be for the developers, stakeholders, and possibly the users that will be supporting this application in the future such as quality assurance teams – this can be further extended to the IT business analysis for further development choices.
- Identify the system/product using any applicable names and/or version numbers – This system/product is going to be designed around the first version 1.0, which will consist of the prototype until the first release.
- References for any other pertinent documents such as:
 - Prerequisite documents – Project Plan, Software Requirements Specification
 - Documents resulting from this document – Documents provided in the future will be the user manual and technical assistance guide for future quality assurance teams.
- Define any important terms, acronyms, or abbreviations.
 - CRUD – Create, Read, Update, Delete
 - UI/UX – User Interface / User Experience
 - Firebase – Google’s all-in-one database solution.

7. System Overview

List/Functionality and Management – Users will be able to create multiple lists that can be updated, read, deleted, and more can be generated.

Collaboration -This system will be able to support real-time updates that will allow all added users to be able to see all the changes made to the list.

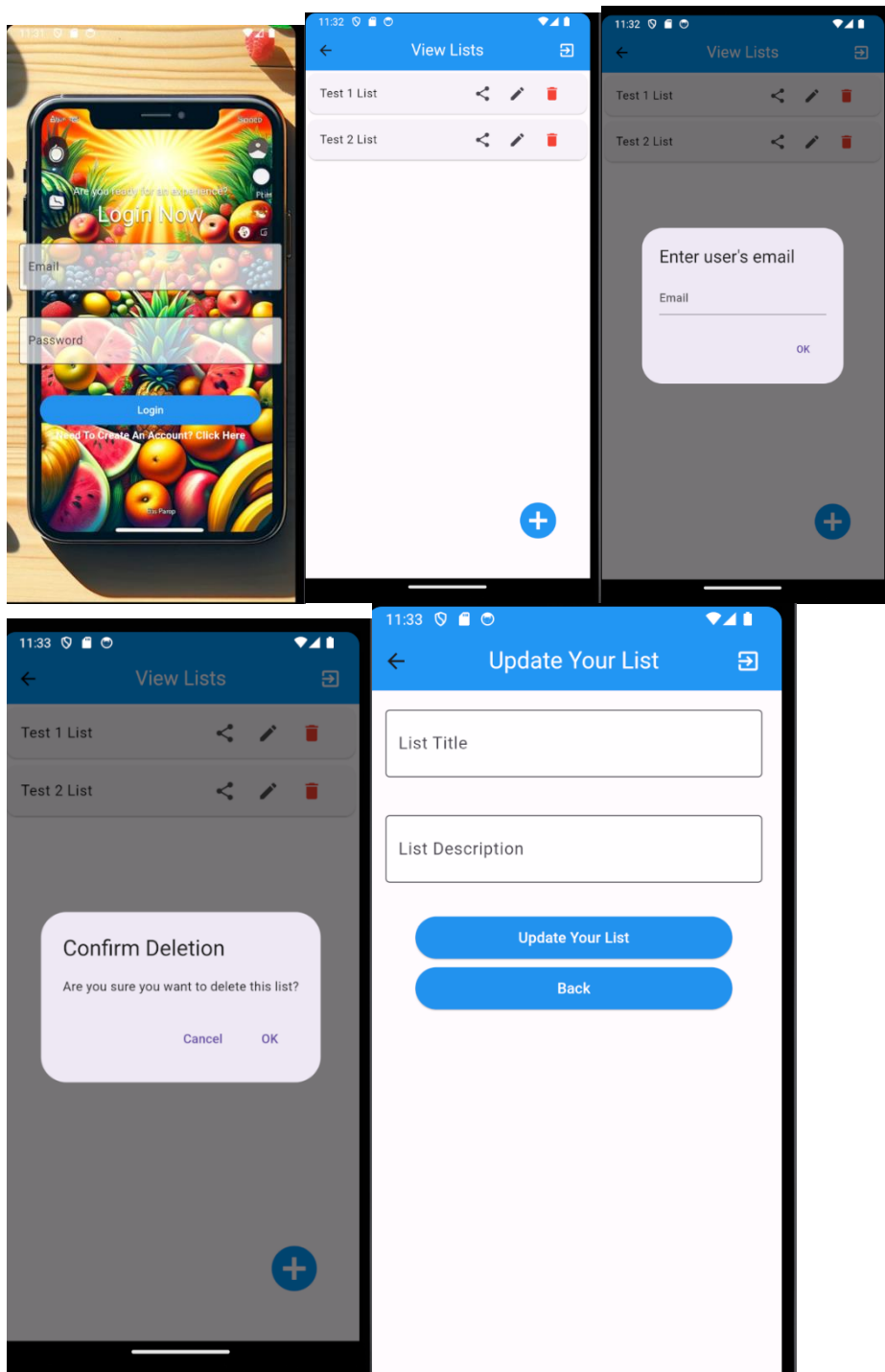
User Account and Authentication – There will be secure registration and authentication that will allow users to create multiple accounts that will hold their independent lists.

Design Approach – Our design will employ more of a microservices approach in this aspect. It will focus on creating each smaller service and bringing the software together based off each independent service created. This will help speed up and simplify the development phase and make sure that we are all on the same page.

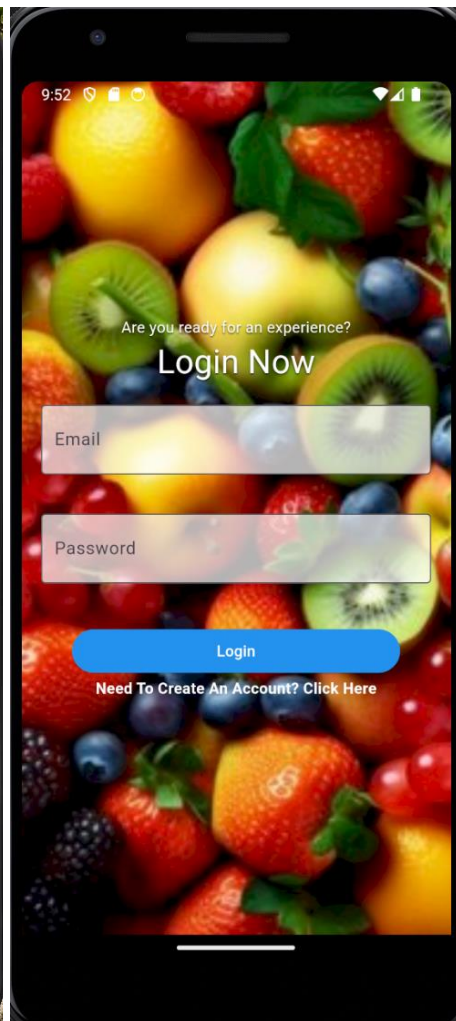
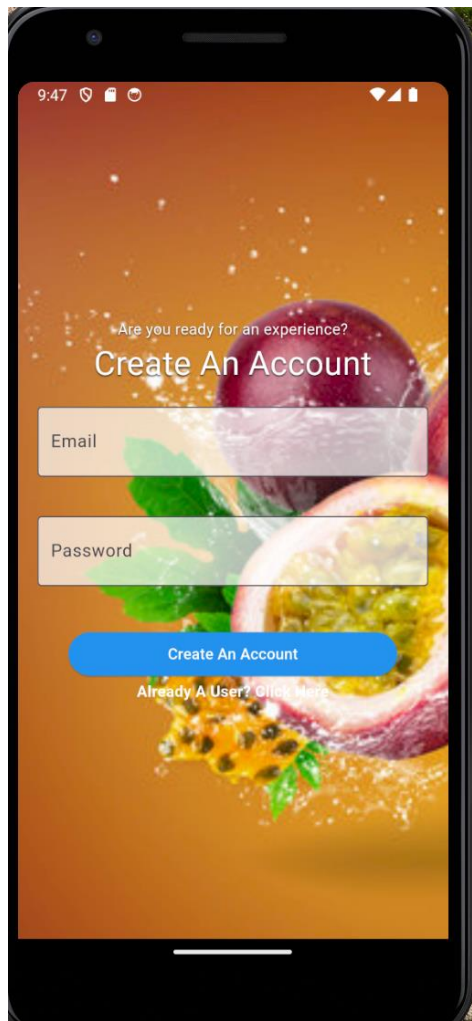
Technology Stack – Our technology stack will stem from the Dart Programming language that will extend into the Flutter Framework to simplify development for not only android but Ios development.

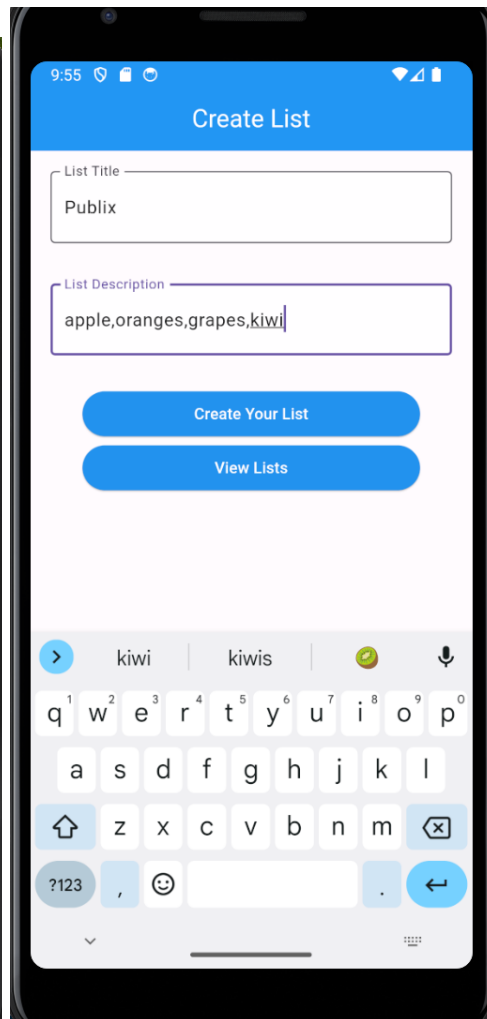
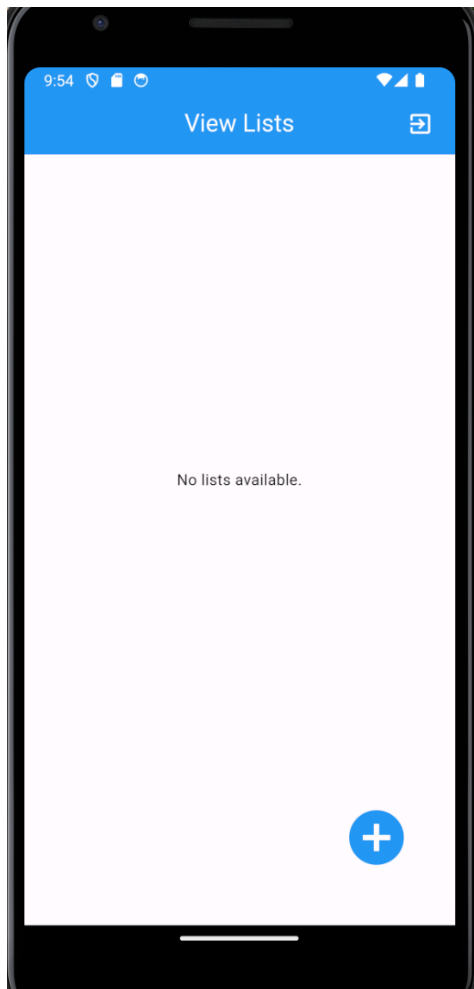
Backend – Our database will be powered by Firebase (back end as a service), and Fire store which is a NoSQL database to simplify store develop with ease.

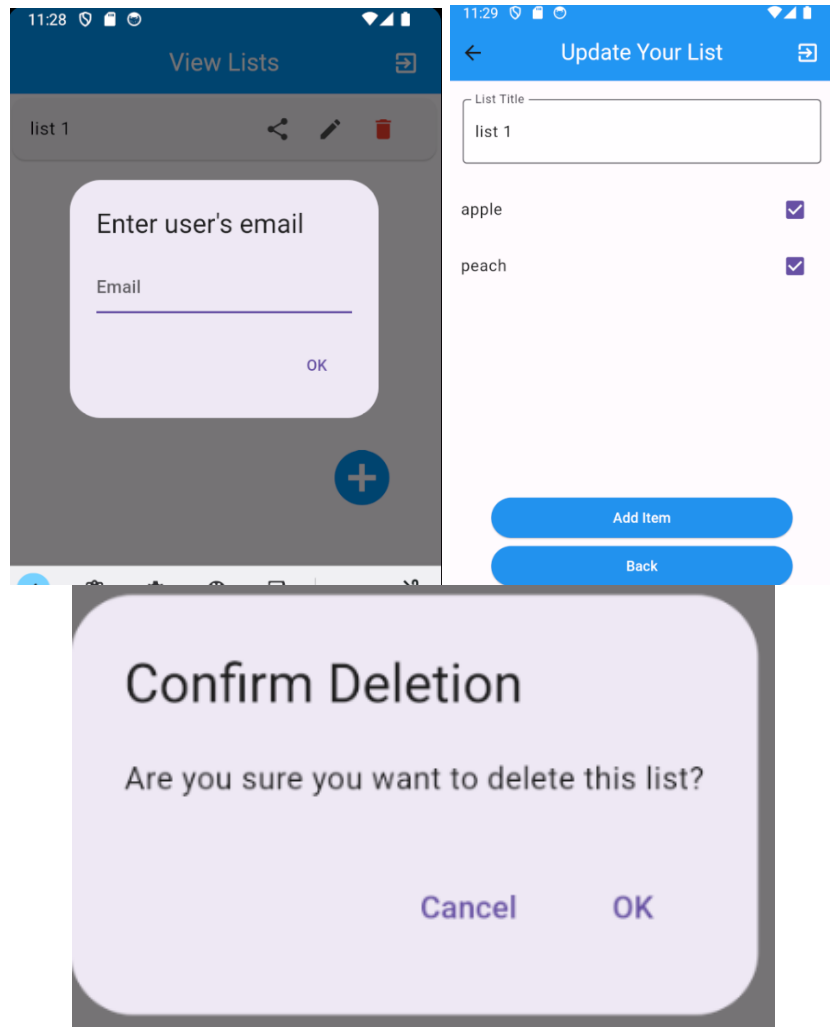
Initial Prototype:



Prototype Phase 2:







8. Design Considerations

This section describes many of the issues which need to be addressed or resolved before attempting to devise a complete design solution.

8.1 Assumptions and Dependencies

- Users have an internet facing device. This software will be dependent on the speed/bandwidth of the device in terms of network bandwidth on a modern device.
- This application is specifically built for and should be used on Ios and Android operating systems compatible on google play, and iOS app store.
- End-user characteristics – end users should have basic technology and detailed literacy skills. They will need to understand how authentication and user accounts as well as the purpose of a real time update database in terms of the grocery list functionality.
- Possible and/or probable changes in functionality – The purpose of this application will remain the same as a dynamic grocery list application that will implement CRUD and variable users per list. In the future, inventory tracking with grocery stores, and grocery list prices could be integrated within this application.

8.2 General Constraints

- Hardware or software environment – This software will need to be applicable to all devices that will have the google play and the iOS app store. It needs to have a solid and variable design that can be applicable to all devices that meet the criteria.
- End-user environment – The environment in which the end-user will be interacting with must have a low learning curve. All symbols and functionality need to be easy to use and pick up for all users with the application. It might need to have offline compatibility.
- Availability or volatility of resources – This application will be dependent upon the firebase and fire store to implement CRUD and the real-time collaboration within the application. Any disturbances would be detrimental and cause a global outage.
- Data repository and distribution requirements – Dependent upon firebase and fire store to handle the distribution and data of all our users and their specified lists.
- Security requirements (or other such regulations) – There will be confidentiality, data integrity, and availability for the user.
- Performance requirements – Users must have and are expected to have a decent internet connection, otherwise this could mean that the data would not be uploaded to the particular and respective list.
- Network communications – The real-time updating is a core function and requires network communications, without the apps core would be removed and non-functional.

- Verification and validation requirements (testing) – Comprehensive testing will be applied to various devices using VMs and physical devices. This will cause a very demanding but rewarding development experience.

8.3 Goals and Guidelines

Describe any goals, guidelines, principles, or priorities which dominate or embody the design of the system's software. Such goals might be:

- The KISS principle ("Keep it simple stupid!")
- Emphasis on speed versus memory use
- working, looking, or "feeling" like an existing product
- Scalability and Flexibility
- Speed and Responsiveness – for users that are on the go inside of the grocery store or need to update very quickly.
- Reliability – If our users expect us to keep their important grocery list items, we must make sure that are app is reliable and secure.
- Maintainability and Testability – Ensure that every feature that is developed has a straightforward testing process for future updates.

8.4 Development Methods

The development method we will follow will be the Agile Methodology. It will consist of project planning, sprint planning, weekly standups (in our case), sprint development, review, and refinement. Throughout each sprint testing, we will concurrently follow as we are developing the next microservice.

- Project Planning – this is the initial stage that we will hold for the project in terms of the thought process and any initial stages to help develop/consider for this application.
- Sprints Planning - Short sprints in our case will be in weeks to get a certain feature/microservice complete.
- Weekly Standups – At the end of each week we all go over what we got done during the sprint.
- Sprint Development – During sprint development we will work as fast and hard as we can to sprint to get each feature complete.
- Review – We will review all completions and sprints and formulate testing through the process to ensure that every feature works.
- Refinement
- Final Launch of project

9. Architectural Strategies

- Dart and Flutter Framework: This ensures that we can work on 2 and one with cross compatibility between Ios and Android development to ensure seamless and low-cost development practices.
- Reuse of existing software components to implement various parts/features of the system – Use of fire store and firebase can be applied to multiple functions of our program. Such as implementing the CRUD to read to distinct parts of the database.
- Future for extending or enhancing the software – We aim to implement added features to implement different prices when you add different items to a list.
- User interface paradigms (or system input and output models) – This will only be a mobile application, so we need to ensure that we have a mobile first design. If web-app or other applications are considered it will be last in the cycle for additions.
- Hardware and/or software interface paradigms – The paradigm that will be used will revolve around a touch-based user interface. There will be a responsive design where the UI will dynamically adjust to different screen sizes within the application. This means that there could be a wide range of devices that have the iOS and android google play store to be able to use are application.
- Error detection and recovery – This will be implemented using the cloud regarding Firebase and Fire store. Since the constraint of this application is that all users must have a valid and solid network connectivity, this will ensure no update/delete issues with real-time collaboration in the application. All error detection and handling on firebase will be used with the Admin Cloud Storage API.
- Memory management policies – Flutter uses client-side caching to store frequently accessed data which can help in reducing the number of network requests. Dart has effective and monitoring when it comes to garbage collection to minimize security and improve performance by reducing the event of a memory leak.
- External databases and/or data storage management and persistence – Data storage management with Fire store is seamless and very intuitive. It provides built-in real-time updates, that will update across all user accounts added to the grocery list. Fire store will cache the data in the event of an outage on the user's device and upload the data in the event when the network outage is resolved. Firebase integrates with Fire store, and has multiple SDKs for iOS Android and the web.
- Generalized approaches to control – Implement feature flags to allow us to toggle certain features in the future in terms of the grocery list pricing interface, which would allow us not to develop any further code.

- Concurrency and synchronization – Optimistic Updates will allow the user to perform an action and right after the action is performed, that action/update will automatically be deployed to their UI but can be processed in the back end.
- Communication mechanisms – WebSocket's for real-time updates: As a second measure we can implement web sockets to create a concurrent two-way communication between the client and the server. This would be fail-safe though as firebase does this flawlessly.
- Management of other resources - Using Firebase cloud functions to run our back-end code, authentication, and HTTP request that may be used to access our future version of the web application.

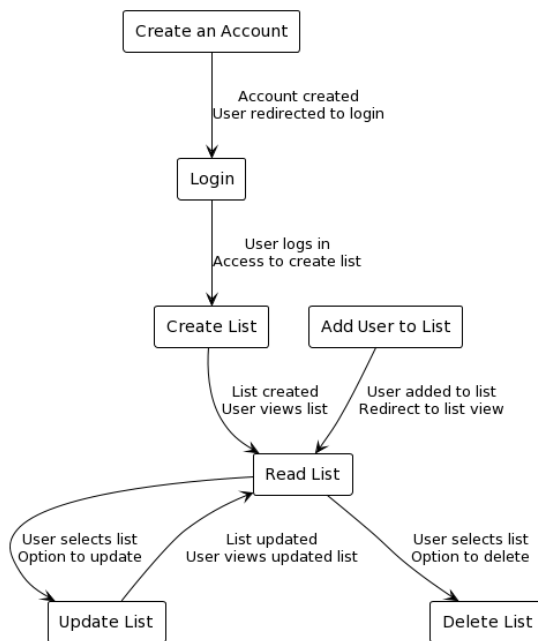
10. System Architecture

This app is divided into two main components, The front end, and the backend. The frontend is made with flutter and focused on a simple user interface that provides a way for users to effectively manage their grocery lists. The design is intuitive and user friendly, making sure that the app looks and works the same whether you access it from your phone or computer.

The backend handles the core data operations of the app such as user authentication, data storage, and synchronization across devices. Fire base plays a pivotal role in our backend processes. It serves as the foundation for our basic operations and its time data feature allows our app to be instantly synched across all devices. This allows us to keep users up to date with one another regardless of how they interact with the app, which greatly improves user experience.

This architecture was chosen for its scalability and reliability. A flutter front end allows us to support multiple platforms such as desktop, android, and windows. And firebase excels at real time data updates and allows for easy scalability to meet increasing demands without compromising performance. This scalability extends to handling a surge in data volume, number of users, and complexity of data operations.

10.1 Subsystem Architecture



Account Creation:

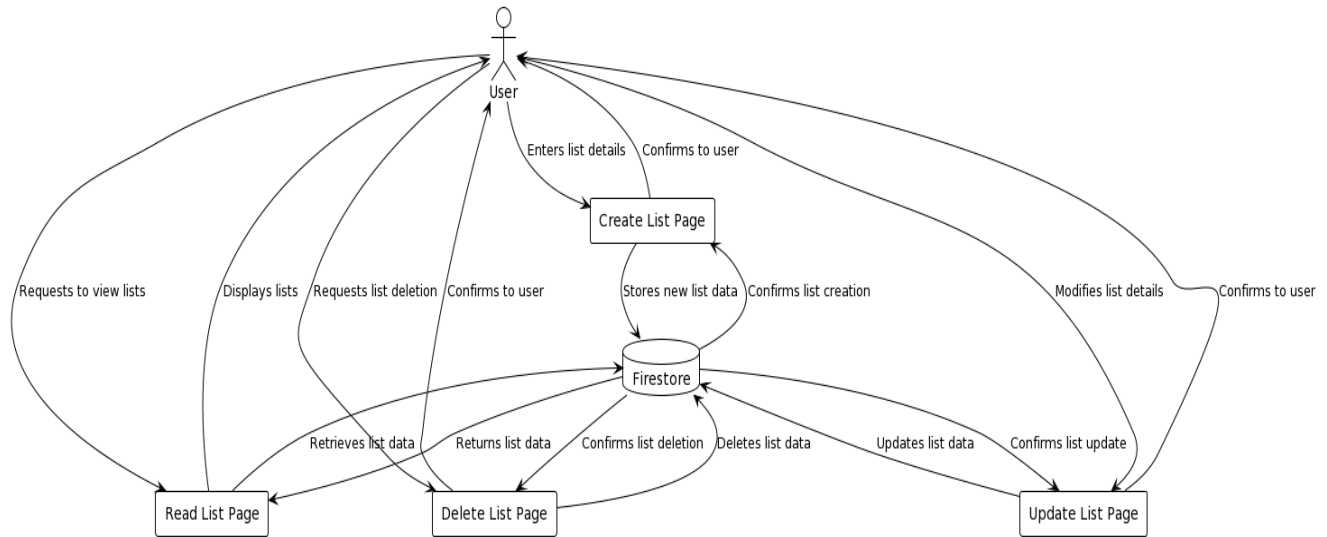
- Users are prompted to create an account then log in.
- After successfully login in a list is created and the user gains access to all list functionalities

List Functionalities:

- Create List - Once logged in, the user can create a new list.
- Add User to List - The user can add another user to a list.
- Read List - The user can view lists they have access to.
- Update List - The user can select a list and choose to update it.
- Delete List - The user also has the option to delete a list.

User Actions:

- Creating a list allows the user to view it
- Adding a user to a list redirect to list view, which allows the user to see.
- Choosing to delete a list will delete the list for that user if it was not originally theirs and delete it across users if it was originally theirs



Page functions:

- Create List page: This is where users can create lists which are then stored in firebase.
- Read List Page: This page allows users to view any lists they have been given access to.
- Update List Page: Allows users to modify their lists.
- Delete List Page: Allows users to delete certain lists they no longer need.

Users:

- Users have the power to create read update and delete grocery lists they own.

Fire store database:

- This serves as the central database of the app where all data management occurs. Here both grocery lists and users are stored.

10.2 Version Control

We used Git for version control during the process of creating our app. Git allowed us to work on different features simultaneously, which greatly improved our speed of development. By ensuring that we consistently pushed and pulled our code changes before working we were able to avoid merge conflicts. Git also proved to be a great tool for tracking the history of our app. If something went wrong we were able to revert to previous versions of our app, identify the problem and fix it. Overall git was a significant part of our development process and allowed us to efficiently collaborate as a team.

11. Analysis

- Choice of which specific product to use (compiler, interpreter, database, library, etc. ...) - Using the Dart Programming Language and the Flutter Framework. This will incorporate and tie Firebase and Fire store together to form a cloud based back-end solution that is very easy to deploy with low environment cost.
- Engineering trade-offs – Some engineering tradeoffs that we considered were between the complexity of the way the UI is designed. In an app like ours it is crucial that the design is formulated to create a user-centered design rather than focusing on the complexities on how everything is processed efficiently.
- Coding guidelines and conventions – Using Flutter and dart, widget constraints must be applied when developing. Leveraging the strong typing that dart features to reduce runtime errors and using a stateful and stateless widget-centered design to re-use and design more UI components.
- The protocol of one or more subsystems, modules, or subroutines – The only applicable protocol is going to be Flutter's 2.0 Navigator for advanced page routing, in addition to using darts exception handling system to manage errors for a great UX.
- The choice of a particular algorithm or programming idiom (or design pattern) to implement portions of the system's functionality – Implementing a list view for all the grocery list that can extend beyond the length of the page. Creating dynamic and stateful widgets that ensure that the state of the page can be changed.
- Plans for ensuring requirements traceability – Our plan to retrace requirements would be that every feature that is being added can be traced back to the requirements documentation using Git by adopting a feature branch workflow – you can further develop this to specific tickets in the corporate environment.
- Plans for testing the software – After every service is developed and finished rigorous testing by us the group members will be tested to ensure that no issues will persist in the first round of the prototype. This will happen during the review and print development phase.
- Plans for maintaining the software – Our plan for maintaining the software will consist of documentation and a README file for the initial release. There will also be regularly updating dependencies that the software is dependent upon such as Flutter SDK to ensure that the security and stability will always be up to the latest date.
- Interfaces for end-users, software, hardware, and communications

- Hierarchical organization of the source code into its physical components (files and directories).
 - FlutterProject
 - .gradle
 - .idea
 - App
 - Gradle
 - Grocerylist <--- Main Project File
 - .dart_tool
 - .idea
 - Android
 - Build
 - Images <--- All Images Related Will Be Here
 - Ios
 - Lib <----- All Pages Will Exist Here
 - Linus
 - Macros
 - Web
 - README <--- Upon Release Instructions Overview Will Be Here
- How to build and/or generate the system's deliverables (how to compile, link, load, etc. ...)
- Download VS Code
 - Install Flutter and Dart
- Download Android Studio
 - Install Emulators (iOS or Android)
- Create Project on VS Code import grocery list from GitHub.
- Run Downloaded emulator from Android.
- Navigate to lib folder in grocery list.
 - Run the main page of the program while the emulator is running.

12. Detailed System Design

12.1 Classification

Flutter Frontend: Subsystem
 Firebase Backend: Subsystem

12.2 Definition

Frontend: The visual and interactive part of the app where users manage grocery lists

Backend: a cloud database that allows for a streamlined way to manage data

12.3 Responsibilities

Frontend: handles user interactions and data display

Backend: responsible for data processing, such as storage authentication and real time synchronization

12.4 Constraints

Frontend: Limited by device capabilities

Backend: 50,000 reads and 20,000 writes a day

12.5 Composition

Frontend:

- Components - This part of the app includes user interface elements such as list views for displaying grocery items, list editors for adding or modifying items, and login screens for user authentication. All these elements are developed using Flutter's widget library.

Backend:

- User Table - The backend houses a user table that stores essential information such as usernames and user IDs, crucial for user management and authentication processes.
-
- Grocery List Table - It features a Grocery List table that details each list's owner, the users it's shared with, and the contents of each list. This table is pivotal for the real-time updating and synchronization of grocery list data across different user devices, facilitated by Firebase's robust data handling capabilities.

12.6 Uses/Interactions

The Frontend and Backend communicate data back and forth between one another. Allow the app to effectively run. Process like data retrieval, user authentication and updates

12.7 Resources

Frontend: internet connectivity and flutter libraries

Backend: cloud storage, computing resources, firebase infrastructure

12.8 Processing

Frontend: Flutter widgets are used to gather user data and this data is then sent to the back end through firebase events

Backend: tables on the backend store the apps grocery lists When a change in one of these tables is detected firebase automatically updates the UI of the flutter app

12.9 Interface/Exports

Frontend: the frontend offers a wide range of Ui components that allow us to provide an interactive interface for our users

Backend: this backend exposes firebases Apis are and allows the app to effectively communicate with firebase to manage data

13. Testing plan

1. Introduction

Purpose: The testing plan outlined below is designed to ensure the quality and reliability of my dynamic grocery list app, which has been developed using Dart, Flutter, and Firebase Fire store. The testing plan covers accessibility, performance, compatibility, security, and user acceptance. By following this comprehensive testing plan, I aim to identify and address any issues or bugs in the app, ensuring that it meets the highest standards.

Scope: The testing will cover all functional and non-functional aspects of the app.

2. Test Objectives

To ensure all features of the app work as intended.

- This includes testing the functionality of all features, such as adding and deleting grocery lists, editing item details, and sharing lists with others.

To verify the app's usability and performance.

- This involves testing the app's user interface to ensure it is intuitive and easy to use. It also includes performance testing to ensure the app is responsive and does not lag or crash under normal usage conditions.

To validate the app's compatibility with different devices and platforms.

- This involves testing the app on a variety of devices, including different smartphones to ensure it works correctly on all screen sizes and resolutions. It also includes testing the app on different operating systems, such as Android and iOS, to ensure it is compatible with both platforms.

To ensure the app's security and data integrity.

- This involves testing the app's security features, such as user authentication and data encryption, to ensure user data is protected. It also involves testing the app's data synchronization with Firebase Fire store to ensure data integrity and consistency across devices.

3. Test Environment

Devices: Android and iOS smartphones.

Internet Connection: Required for syncing.

4. Test Scenarios

1. User Registration and Login

Verify that users can register an account. **Pass** | Fail

Verify that users can log in using their registered credentials. **Pass** | Fail

2. Creating a Grocery List

Verify that users can create a new grocery list. **Pass** | Fail

Verify that users can add items to the list. **Pass** | Fail

Verify that users can remove the list. **Pass** | Fail

Verify that users can edit item details. **Pass** | Fail

3. Managing Grocery Lists

Verify that users can view their existing grocery lists. **Pass** | Fail

Verify that users can delete a grocery list. **Pass** | Fail

Verify that users can archive a grocery list. **Pass** | Fail

4. Sync and Cloud Storage

Verify that changes made to a grocery list are synced across devices. **Pass** | Fail

Verify that data is securely stored and retrieved from the database. **Pass** | Fail

5. Performance and Load Testing

Test the app's performance with many items in a list.

Pass | Fail

Test the app's response time under various network conditions.

Pass | Fail

6. Usability and Accessibility

Verify that the app is easy to use and navigate.

Pass | Fail

Verify that the app complies with accessibility standards.

Pass | Fail

7. Security

Verify that user data is encrypted and securely stored.

Pass | Fail

Verify that the app protects against common security vulnerabilities.

Pass | Fail

5. Test Execution

Testers will perform the test scenarios outlined in this document.

Test cases will be executed manually on various devices and platforms.

Automated testing tools may be used for regression testing and performance testing.

14. Test Report

14.1 User Registration and Login

The user registration process allows new users to create an account by providing their email address and creating a password.

The login functionality allows users to enter their registered email address and password to access the app. Successful login grants users access to their grocery lists.

14.2 Creating a Grocery List

Users can easily create a new grocery list by clicking on the "New List" button and providing a name for the list. The app then creates a new empty list for the user.

Adding items to the list is straightforward, with users able to enter different items names separated by coma. The app provides a simple interface for this process.

Users can edit items by selecting a pen icon next to each list they created.

Users can share their lists with other app users, enabling collaboration and easy sharing of shopping lists.

14.3 Managing Grocery Lists

Users can easily view their existing grocery lists from the main screen of the app. The lists are displayed in a clear and organized manner, making it easy for users to find the list they want.

Deleting a grocery list is a simple process, user can swap to the right to delete the list they've created.

14.4 Sync and Cloud Storage

The app's sync feature ensures that changes made to a grocery list on one device are reflected on all other devices where the user is logged in. This ensures that users have access to their most up-to-date lists no matter which device they are using.

Data is securely stored and retrieved from the database using Firebase Fire store, which provides a reliable and scalable solution for storing app data.

14.5 Performance and Load Testing

The app performs well even with many items in a list, displaying smooth scrolling and responsive interactions.

The app's response time is consistently fast, even under various network conditions such as low bandwidth or intermittent connectivity.

14.6 Usability and Accessibility

The app's user interface is designed with usability in mind, featuring clear navigation and intuitive controls. Users can easily navigate between different sections of the app and perform tasks without confusion.

The app complies with accessibility standards, making it usable for users with disabilities. For example, the app works well with screen readers, providing a text-based description of elements for users who are visually impaired.

14.7 Security

User data is encrypted both in transit and at rest, ensuring that it remains secure from unauthorized access.

15. Conclusion

In conclusion, the development of our dynamic grocery list app has been a rewarding journey, showcasing the power and flexibility of Dart, Flutter, and Firebase Fire store. Our team's dedication to meticulous planning and execution has resulted in the successful creation of a highly functional and intuitive application. This app simplifies the task of managing grocery lists.

15.1 Key Achievements

Developed a comprehensive requirement document that served as the foundation for the entire project, ensuring clarity and alignment with app expectations.

Designed an intuitive and visually appealing user interface that enhances the user experience and promotes efficient list management.

Implemented Firebase Fire store for secure and scalable data storage, enabling real-time synchronization across devices.

Created a thorough testing plan and executed various test cases to ensure the app's functionality, usability, and reliability.

15.2 Lessons Learned

Collaboration and communication are essential for project success. Regular meetings and updates kept the team aligned and motivated.

Flexibility and adaptability are crucial in the face of challenges. We encountered some obstacles but managed to overcome them through innovative solutions and teamwork.

Overall, the development of our dynamic grocery list app has been a rewarding experience that has allowed us to grow as developers. It has served as a platform for our team to expand our skills, both individually and collaboratively, and has provided us with the opportunity to deliver a product that we are genuinely proud of. We have faced and overcome various challenges, which has not only strengthened our technical abilities but also our problem-solving and teamwork skills.

16. Appendix

16.1 Training

All users of our group completed the flutter tutorial and pasted the results of their code in their emulator. The final app generates random text for each run of the code. The last page of the tutorial was reached for group members. A screenshot was taken and submitted to team lead (Lee Martin) to be posted in this training section.

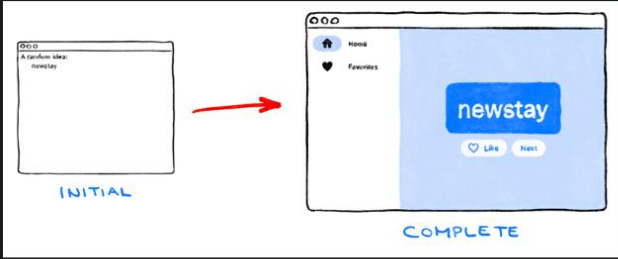
The last page of the tutorial is linked here:

<https://codelabs.developers.google.com/codelabs/flutter-codelab-first#8>

9. Next steps

Congratulations!

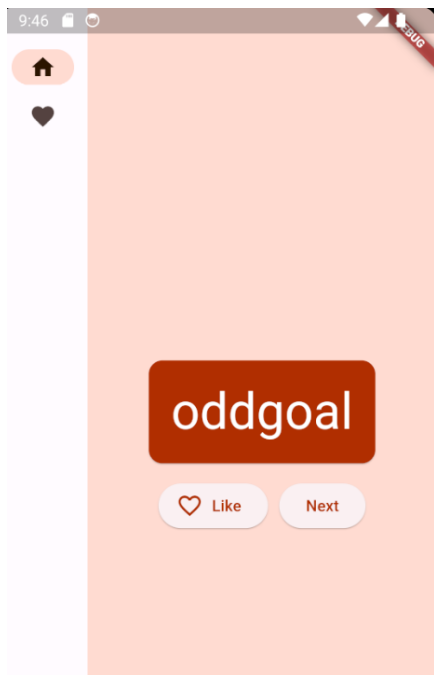
Look at you! You took a non-functional scaffold with a `Column` and two `Text` widgets, and made it into a responsive, delightful little app.



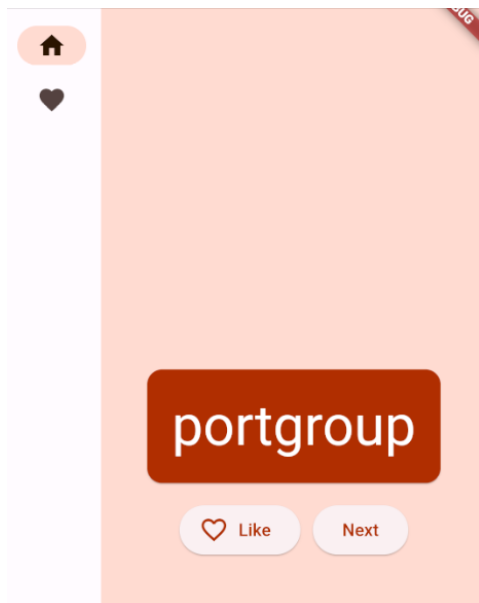
What we've covered

- ✓ The basics of how Flutter works
- ✓ Creating layouts in Flutter
- ✓ Connecting user interactions (like button presses) to app behavior
- ✓ Keeping your Flutter code organized
- ✓ Making your app responsive
- ✓ Achieving a consistent look & feel of your app

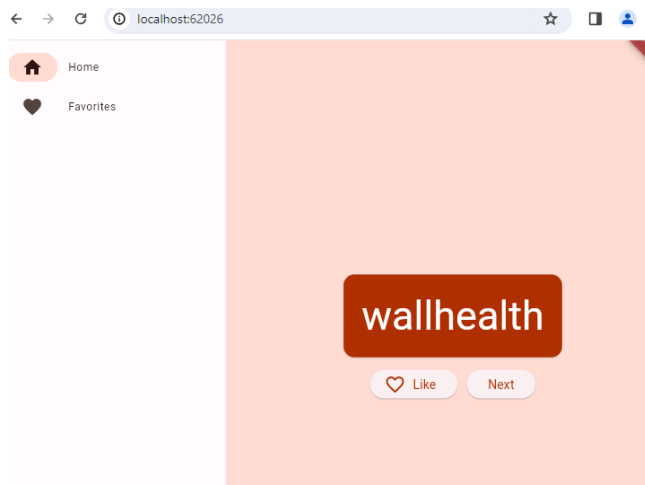
Lee Martin



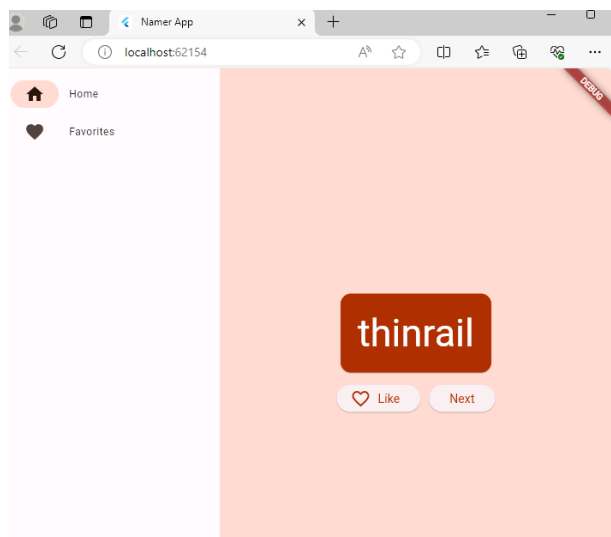
Richard Thomas



Jesse Schultheis



Yesakh Ergano



17. Glossary

Flutter – open-source framework that is particularly used by Dart language. It is an open-source software development kit.

Dart - a programming language designed by Lars Bak and Kasper Lund for google. It is used for low-overhead and fast development can be applicable with Flutter for development with iOS and Android.

UI – short for user interface

UX – short for user experience.

Fire store - a NoSQL document database that lets you easily store, sync, and query data for your mobile and web apps - at global scale.

Firebase - is a cloud-hosted NoSQL database that lets you store and sync data between your users in Realtime.

18. Bibliography

Development. Flutter. (n.d.). <https://flutter.dev/development>

Google. (n.d.-a). *Handle errors for cloud storage on web | cloud storage for firebase.*

Google. <https://firebase.google.com/docs/storage/web/handle-errors>

Google. (n.d.-b). *Handle errors for cloud storage on web | cloud storage for firebase.*

Google. <https://firebase.google.com/docs/storage/web/handle-errors>

State diagram syntax and features. PlantUML.com. (n.d.). <https://plantuml.com/state-diagram>