A Software Design Specification: SP-5 Orange Grocery List – Software Design Document (SDD)

Course CS: 4850

Semester: Spring 2024

Professor: Sharon Perry

Date: 02/05/2024

Team: SP-5-Orange- Dynamic Grocery List

 Lee Martin	 Richard Thomas
 Jesse Schulteis	 Yesahk Ergano

- **Project Team**

Roles	Name	Major responsibilities	Cell Phone / Alt Email
Project owner	Faculty Member	N/A	N/A
Team leader	Lee Martin	Documentation/Developer	470-774-6074 Leemartin3232@icloud.com
Team members			
	Richard Thomas	Developer	404-932-3120 RThomas1137@gmail.com
	Yesahk Ergano	Test	678-599-3598 Yesahkergano1272@gmail.com
	Jesse Schulteis	Developer	678-687-3102 Schulteis360@gmail.com
Advisor / Instructor	Sharon Perry	Facilitate project progress; advise on project planning and management.	770-329-3895

Table of Contents

1. Introduction

1.1. Document Outline

- Introduction
- System Overview
- Design Considerations
    - Assumptions and Dependencies
    - General Constraints
    - Goals and Guidelines
    - Development Methods
- Architectural Strategies
- System Architecture
- Policies and Tactics
- Detailed System Design
- Glossary
- Bibliography

1.2. 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.

1.2.1. Introduction

Provide an overview of the entire 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 realease.
- 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.

1.2.2. System Overview

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

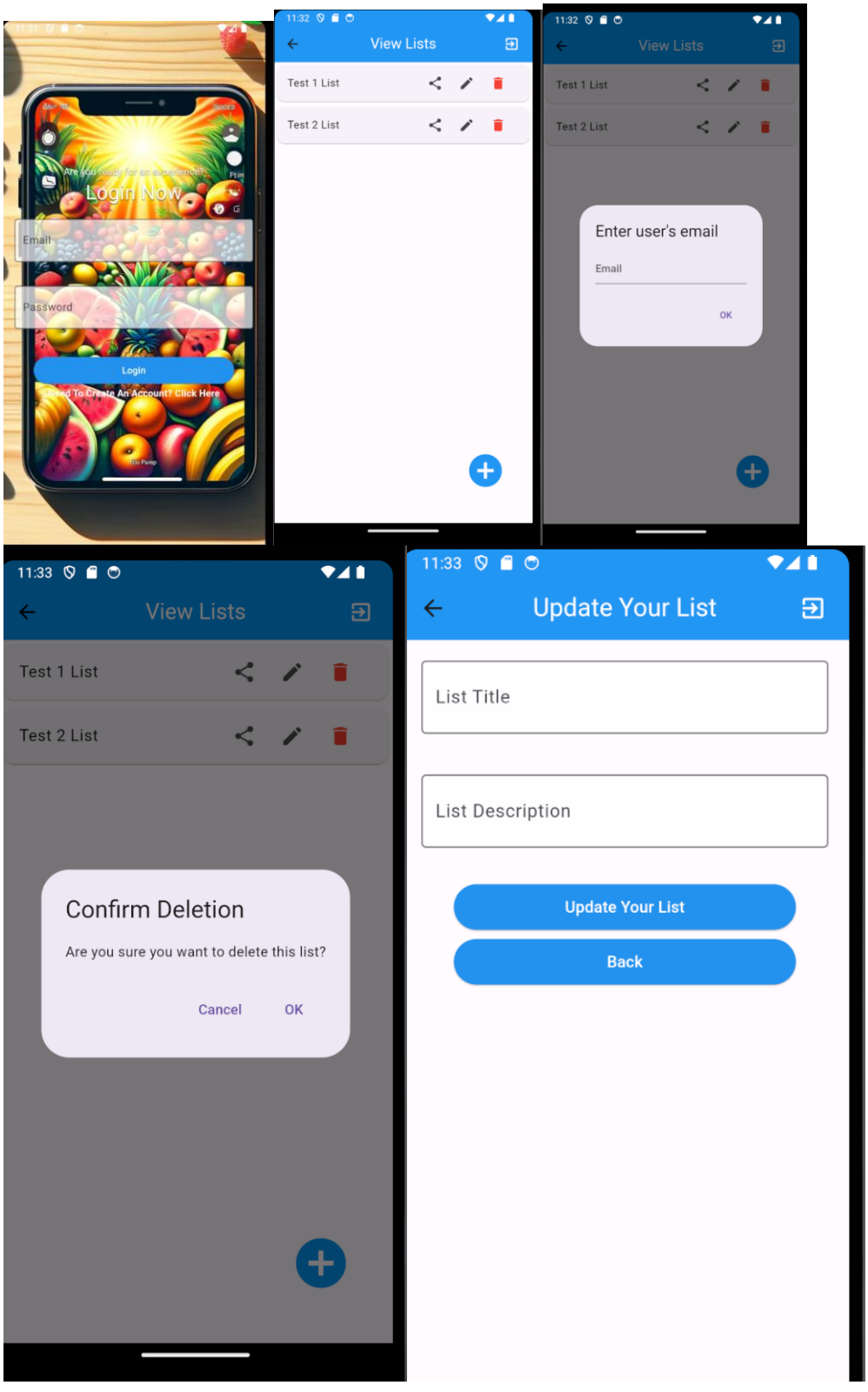
Collaboration -This system will be able to support real-time updates that will allow all added users to be able to see all the changes made to the list.

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 of 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 Firestore which is a NoSQL database to simplify store develop with ease.


Login Now
Email
Password
Login
Need To Create An Account? Click Here
11:32
View Lists
Test 1 List
Test 2 List
Enter user's email
Email
OK
11:33
Confirm Deletion
Are you sure you want to delete this list?
Cancel
OK
Update Your List
List Title
List Description
Update Your List
Back

2. Design Considerations

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

2.1. Assumptions and Dependencies

Describe any assumptions or dependencies regarding the software and its use. These may concern such issues as:

- 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 ditial 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.

2.2. 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-urser 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 firestore in order 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 firestore to handle the distribution and data of all of 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.

2.3. 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 ensures
- Maintainability and Testability – Ensure that every feature that is developed has a straightforward testing process for future updates.

2.4. 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

3. 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 firestore 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 Firestore. 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 freqeuntly 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 Firestore 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 users device and upload the data in the event when the network outage is resolved. Firebase integrates with Firestore, and also has multiple SDK's 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 – Websockets 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 firebases do 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.

4. System Architecture

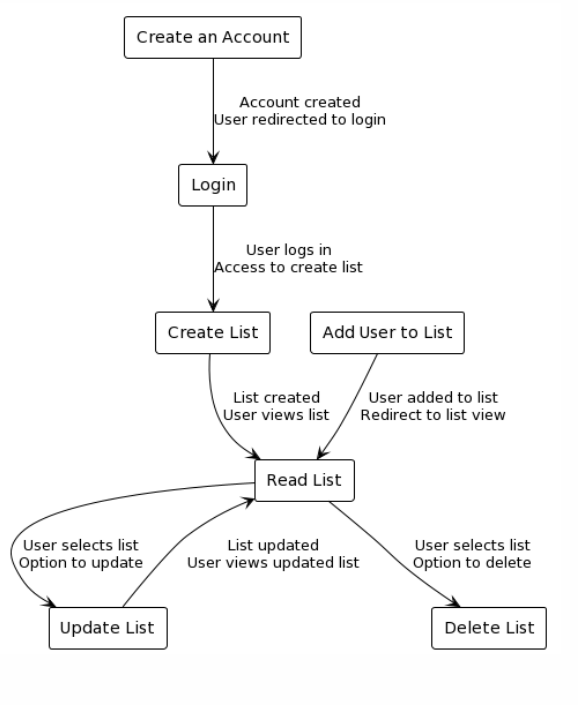
This app is divided into two main components, The front end and the backend. The frontend is made with flutter and focused on a simple user interface that provides a way for users to effectively manage their grocery lists. The design is intuitive and user friendly, making sure that the app looks and 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.

4.1. Subsystem Architecture



Account Creation:

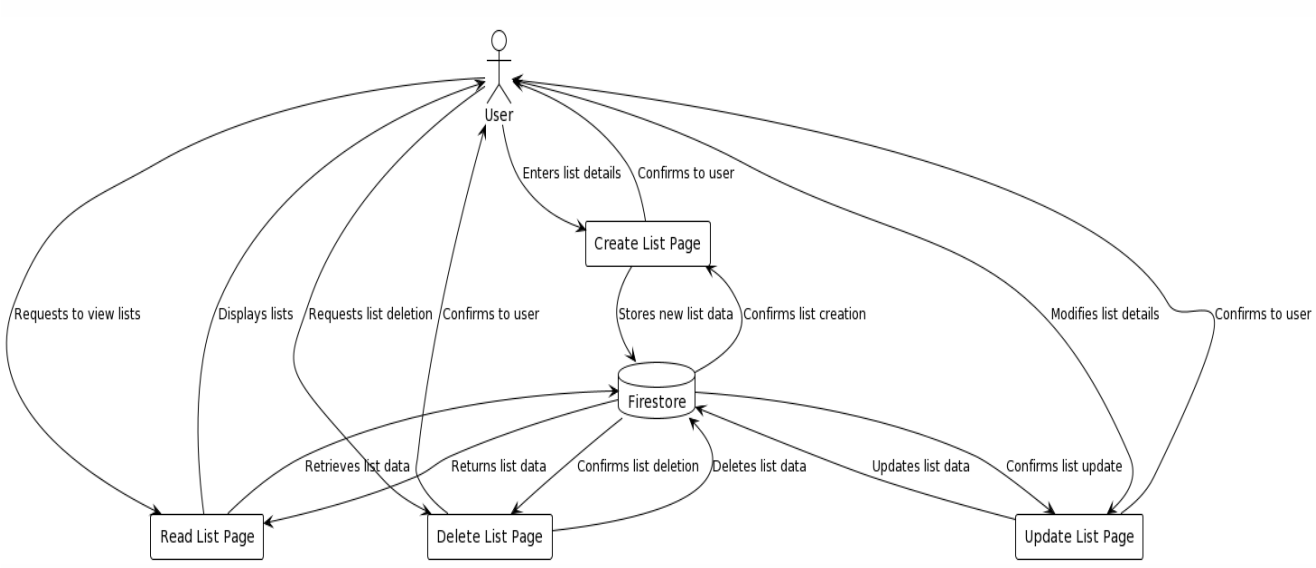
- Users are prompted to create an account then log in
- After successfully login in a list is created and the user gains access to all list functionalities

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

Firestore database:

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

5. Policies and Tactics

- 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 Firestore 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 of 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 corperate encironment.

- Plans for testing the software – After every service is developed and finished rigerous 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 grocerylist from github
- Run Downloaded emulator from Android
- Navigate to lib folder in grocerylist
    - Run the main page of the program while the emulator is running.

6. Detailed System Design

6.1. Classification

Flutter Frontend: Subsystem
Firebase Backend: Subsystem

6.2. Definition

Frontend: The visual and interactive part of the app where uses mange grocery lists
Backend: a cloud database that allows for a streamlined way to manage data

6.3. Responsibilities

Frontend: handles user interactions and data display
Backend: responsible for data processing, such as storage authentication and real time synchronization

6.4. Constraints

Frontend: Limited by device capabilities
Backend: 50,000 reads and 20,000 writes a day

6.5. 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.

6.6. 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

6.7. Resources

Frontend: internet connectivity and flutter libraries

Backend: cloud storage, computing resources, firebase infrastructure

6.8. 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

6.9. Interface/Exports

Frontend: the fronend 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 in order to manage data

7. Glossary

Flutter – open-source framework that is particularlly 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.

Firestore - 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.

8. 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