CS4850-03 Fall 2023

Sharon Perry

November 23, 2023

LinkList Final Report

For

SP5-Red Dynamic Grocery List Application

https://sp-5-red.github.io/website/ ***https://github.com/SP-5-Red/LinkList***

Prepared by:

Aly Hussein

Mohamed Shaaban

Joel Juarez

Jeff Williams

Table of Contents

1.0 Development Process

1.1 General Overview

The development process for the Grocery Application is characterized by an iterative and user-centric approach, emphasizing collaboration, adaptability, and continuous improvement. We begin with a comprehensive analysis of user needs, industry trends, and technological possibilities. This process informs the creation of a detailed project plan, outlining milestones, deliverables, and timelines.Continuous integration and automated testing ensure the robustness and reliability of each iteration. Throughout development, user experience (UX) and user interface (UI) design are integral, incorporating design thinking principles to create an intuitive and visually appealing application.

1.2 SDLC

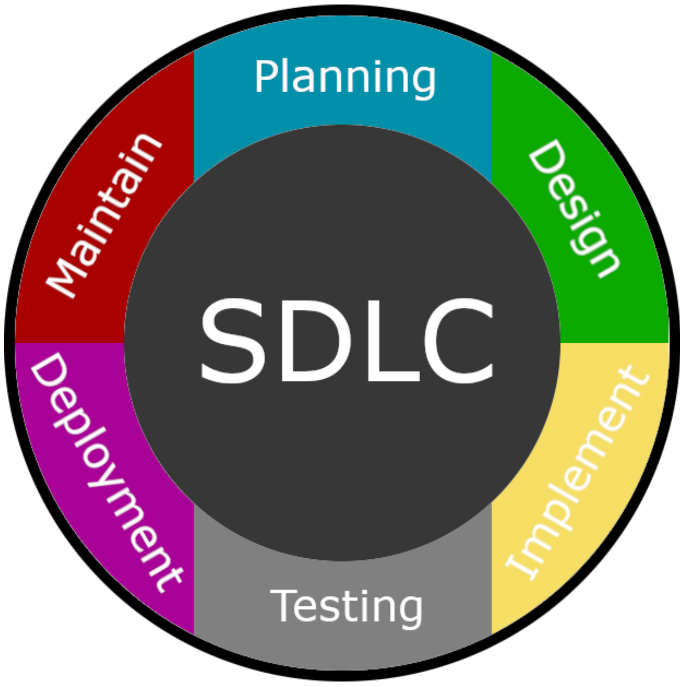


Figure 1: SDLC diagram

1.2.1 Planning

This phase involved defining the purpose and goals we wanted our grocery app to have. It included figuring out what the needs and wants of our users are, then planning the resources required for the project accordingly.

- Define the target audience for the app.
- Determine what key features we need to have.
- Create a project plan with timelines.

1.2.2 Design

This phase is characterized by the approach of creating an intuitive and visually appealing user interface. Starting with a clear analysis of the user requirements and expectations, this informs the creation of the blueprint for the application's architecture. The collaboration between designers and developers is integral, fostering a cohesive approach that prioritizes both functionally and visually. Continuous iterations in design, user testing, and feedback loops can refine the application's design, which will result in a visually polished and user-centric interface that enhances the overall user experiences.

1.2.3 Implementation

This phase is where the development happens.

- Set up the development environment (Android Studio).
- Develop the app while following the design documents and timelines.
- Write unit tests alongside development to make sure that each part functions correctly.

1.2.4 Testing

This phase is where any bugs or issues are found within the code. Although, this phase was a continuous phase throughout the implementation phase. Developers constantly checked their code and tested it throughout the two previous phases to ensure an optimized and efficient application without any errors. It was also tested with other implementations added by developers to ensure a collaborative program between all front-end screens and their backend counterparts.

1.2.5 Deployment

This phase starts once the app is tested and ready for release.

- Deploy the app and make it available for users to use.
- Ensure that all operational best practices are followed.
- Prepare to rollback just in case any issue occurs.

1.2.6 Maintenance

This phase occurs post deployment, the app will need regular maintenance and updates to insure everything operates smoothly.

- Monitor the app performance and feedback to ensure all bugs/issues are fixed.
- Plan for future versions with enhancements and new features.

1.3 Possible Expansions

There are many possible expansions for this newly designed app but the current plans are listed below.

- API integration with retail services for price and availability checks/comparisons
- More customization options (more colors and implementation of fonts)
- Budgeting charts built in to reports
- Budget notifications when nearing limit
- Allow users to make a custom budget
- Different types and an increase in grocery items in program
-

2.0 Version Control

Utilizing a distributed version control system, such as Git, allows us, as developers, to work collaboratively on the same codebase without conflicts. Each iteration, or "commit," is documented with a clear message describing the changes made, facilitating transparency and traceability. Branching strategies enable the creation of separate development paths for specific features or bug fixes, preventing disruption to the main codebase.

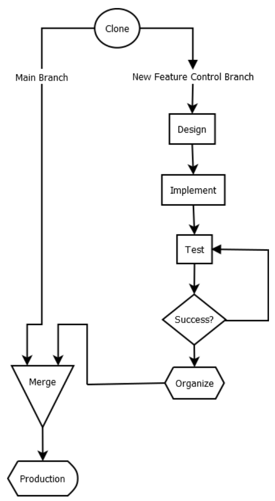


Figure 2: Git version control plan

3.0 Information Flow

The information flow within the Grocery Application is meticulously designed to foster seamless communication and collaboration among members.Continuous integration practices enable real-time updates on code changes, fostering transparency and rapid collaboration. Automated testing and quality assurance processes provide feedback loops to developers, ensuring that the application meets predefined standards before deployment.

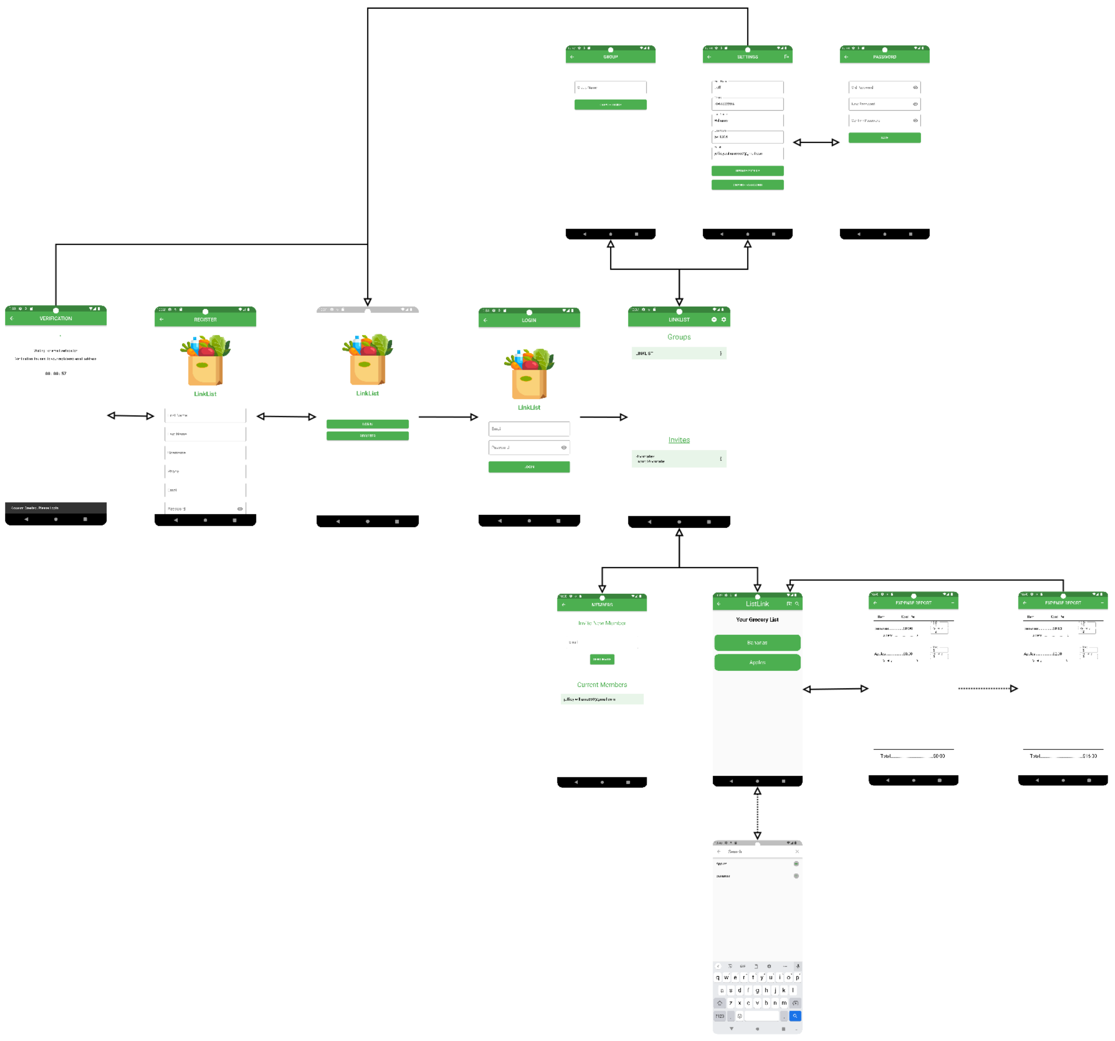


Figure 3: Prototype information flow

4.0 Project Narration

4.1 General Narration

- Planning phase- involves defining the purpose and goals that we wanted the grocery application to have. It includes figuring out what the needs and wants of the users are. We then plan the resources required for the project accordingly
- Design phase- this is characterized by the approach of creating a visually appealing and intuitive user interface. This starts with a clear analysis of the user requirements and their expectations. This will lead to the creation of the blueprint for the application's architecture
- Implementation phase- where the development happens. We first set up the development environment (Android studio for this application). Then we develop the app while following the design documents and timelines. Finally we write unit tests alongside development to make sure that each part functions correctly.
- Application testing- this phase is where bugs and issues can potentially be found in the code. This phase was continuous throughout the implementation. Developers constantly check their code and test it throughout the previous two phases in order to make sure the optimized and efficient application has no errors.
- Deployment- this phase begins once the application is tested and ready for release. We first deploy the app and make it available for all users to use. Then we ensure that all operational best practices are followed closely. Finally we prepare to rollback in case any issues or errors occur.
- Maintenance- this phase starts after the deployment, the application will require regular maintenance and updates to ensure that everything is running smoothly. We do this by monitoring the app performance and looking at feedback to make sure all bugs or issues are resolved. We then also plan for future versions with new enhancements and features.

4.2 Presentation Transcription

Aly Hussein

Hey everyone.
This is Ali Hussein, part of the SP5 Red Grocery List app, a group, and we're in a CS 4850 senior project and the third section for the fall. And our professor is Sharon Perry. And so the group members are me, Aly Hussein, Mohammed Shaaban, Jeff Williams, and Joel Juarez, and we made the grocery list application called LinkList.

Alright so the programming language we used for this application was the DART programming language using Flutter and for this language it's possible to write a single code base for multiple platforms. Well, and this allows us to use to be able to use Android, iOS, web applications and *even desktop applications. And one of the unique features in this application is that it has plugins that lets us have access to cameras, the microphones, or the location services of the device using APIs and also a really positive thing about this application is that it's open source, which means we can have a wide range of resources and third party packages that we can always use available to our tool belt.*

And the IDE that we used for this application was Android studio and this this IDE is specifically made for making Android applications and it's pretty cool cause that it has a built-in emulator to test these applications on different sizes or like different devices as well. And it's really simple to use because it has the drag and drop layout editor and this this helps simplify the designs for the visual element. And the best thing about it is also that it integrates seamlessly with Firebase database.
And the thing about Firebase, the database that we used for this application. It's a cloud hosted database which allows developers to synchronize and store data and in real time and it's very comprehensive with mobile and web applications and desktop as well. And the best thing about it is that it offers real time synchronizations and we can use different clients for this with whether it be mobile or web and it connects to the database which is best for applications that will require constant updates such as such as this application right here and the user data will be stored in the cloud.

So, for our SDLC process. We'll start off with the planning phase and this phase involves defining the purpose and goals that we wanted our grocery app to have and includes figuring out what the needs and wants of our users are.
Then we plan the resources required for the project accordingly.
Then the design phase is characterized by the approach of creating an intuitive and visually appealing user interface. We start with a clear analysis of the user requirements and expectations. This also this ends up informing us with the creation of the blueprint for the application architecture, Then the implementation phase is where the actual development happens. We first set up the development environment and for this application we use the Android studio and then we developed the application while following the design documents and the timelines and finally we write unit tests alongside the development to make sure that each part functions the right way correctly.

Mohamed Shaaban
Hi this is Mohamed Shaaban and I'll be speaking about the application testing. This phase is where any bugs or issues are found within the code. Although this phase was a continuous phase throughout the implementation phase, developers are constantly checked their code and tested it throughout the two previous phases mentioned to ensure an optimized and efficient application without any errors.

Joel Juarez Vazquez

Alright this is Joel Juarez. I'll be talking about deployment. This is the phase that starts once the app is tested and ready for release on an App Store. What happens is that, uh, deploy the app and make it available for the users and ensure that all operations are followed to the best practices, and be prepared to rollback just in case there are any major errors.

Jeff Williams

Hi, my name is Jeff Williams and I'll be covering the maintenance phase within their software development lifecycle. So, this phase occurs post deployment. The app needs regular maintenance and updates to ensure everything operates smoothly. We will monitor the app's performance and any feedback we are given to ensure all bugs and issues are fixed as soon as possible. And we will plan for future versions with enhancements and new features as listed in our final report.

Mohamed Shaaban

So we'll basically be going over the main functions for each of the pages of the app.
I'll start off with the launch screen. For the launch screen, the checkLogin function determines if a user is already logged in by checking the logger storage for the user's email. If the email is found that the user is directed to the apps Home page, home screen, otherwise, they're navigated to the login screen.
For the register screen, the changePasswordIcon function pretty much just toggles the password visibility. The isEmail function checks email viability using a regex and the createAccountFunction register users via Firebase stating details to Firestore and sends a verification email before redirecting them to the verification screen.

For the verification screen, the optTimeOut function enables a resend verification link button and the checkEmail verified function confirms whether the email verification link has been clicked.

Once the verification email has been clicked, then the user can log in.
Uh, can log in via the log in screen. The isEmail function checks an email against a regex and returns true uh if it matches, false otherwise. The signInUser authenticates users through Firebase authentication and if successful, navigates them to their home screen.

For the home screen, the leaveGroup function allows the user to exit a group using the document ID and the email address of the logged in user, and if there are no more members in the group, it would remove the group from the database. The isExpired function checks if any invites related to the users expired based on the expiration date set at the creation of the invite. The removeInvite function removes an invite from the database. For both the acceptInvite and declineInvite function, either add a member to the group's members list and then it deletes the invite, *in that case the user has accepted, or deletes the invite from the system, and in that case the user has declined.*

For the create group screen. The checkForm function establishes that the text box is not empty and the create function establishes a group in the Firebase Firestore data base using the above parameters and adds the user that created the group to the members list.

Joel Juarez Vazquez
Alright, and then on the settings screen, we have the updateProfile method which updates the user details from what is listed above. We have the checkEmail verified method which ensures email verification when changed and checks with the verification link has been clicked. Finally, the update group email function is triggered when an email is updated, ensuring associated groups transition from the previous email to the new one.

Jeff Williams
Alright, so moving on to the change password screen. The checkForm method validates that all form data is populated and the updatePassword method first reauthenticates the user and then updates the password in Firebase authentication.

Now a new edition is the customization screen, so isThemeValue returns if the theme is either in light mode or dark mode based on the inputted integer value. isColorValue returns if the selected color is not currently selected. setThemeValue sets the theme value to either light or dark for preview purposes. setColorValue sets the color value to inputted variable for preview purposes and saveSelection just saves the current theme selections to the hardware storage and the cloud storage and then exits to the home screen.

Moving on to the theme controller, this is what operates the coloring in our app, so getTheme just gets the theme that is stored in the cloud for both preview and storage purposes. SetTheme sets the theme in the cloud to the current selection. getPreviewColor returns the color and theme for the customization screen only based on its current theme values. getColor returns the color for the whole app based on current theme values.

Now the invite new member screen in our groups. So the checkForm function checks if the e-mail entered is a valid e-mail, then checks if the emails within the existing collection user collection, then checks if the users within the group already. The addToGroup function creates an invitation with the above parameters and adds it to the database the expiration date is exactly 30 days after the creation of the invite

Joel Juarez
Alright, and then we have the cart screen. What the cart screen does is that from the home screen, a document ID is passed into the grocery list or the cart screen.
And from there we use that ID to locate the group and find the cart item inside of each group which is used, which is just a list of strings that's stored on Firebase.

And then we actually search for an item. We layer a search widget on top of the main screen and when you start typing it builds a new list of items based on the on what's on the search bar, and then there's a successful search which is a is built and makes a list of all the items that match the search.

And in order to remove items from the cart, there's, uh, well first you have to add the items, so in order to add an item you just click on the little plus arrow that's on the search widget and then when you want to remove, there's two ways there's check, which basically says that you actually did purchase the item, which is information that can be used later. And there's delete, which just completely wipes the item from Firebase since we don't have anything to do with it, that's for if you add something on accident, you don't have to check it so it doesn't mess up some future screens that we're gonna talk about.

And right here is the implementation on Firebase. So, as you can see on one side we have bread when it's added on there and then after that it's deleted on Firebase.

Jeff Williams
Alright so, moving on to the report screen. The getFontSize function returns the font size for one of the three types, item, subnote or footer. If the type doesn't match any established type, it returns a font size too small to see with an eye. The getStringWidth function creates a TextPainter using the provided string and type, then returns the pixel width of the screen or string. setItemList take in the inputs from the text boxes and sets the lists in the database to match, if the text boxes are populated that is. The getPrice function returns the user entered total price divided by the user entered amount. If there's no listed quantity it returns 0. The formatReportScreen function uses the string pixel width to determine how many ellipses to design the string properly and returns the correctly formatted string. getPurchaseTotal function returns the sum of all the prices within the groupPrice array which is entered by the user.

That's the end of our presentation and now we will move on to the demo portion of it.

Joel Juarez

Alright, so this is a demonstration of the Link List app for groceries, so we can start here on the Register screen. And here we actually have to click register to make a new account. And we just basically enter all our information in, so I'll demonstrate right here. Alright, so as you can see, once we make the account we have a certain amount of time to be able to verify our e-mail. To do that basically just go to our e-mail. Click on the link it tells us to click. And now we're verified so scroll back over here. And we can now log in using the information that we used. Right now, we're here on this screen. And we have our groups and our invites. And if we want to invite or we want to create a group, we just click this. We'll create one called roommates. Alright, as you can see, we created this group and now we can basically just invite others. So go to members, you can send someone an invite for example. I'll send it to one of my other accounts right now. So you can see the invite has been sent. Now the other person just has to accept that invite. Alright, so now let's say we actually want to add something to our cart, right? So we would open our cart. So you can see initially it's blank, you have to look for something. So let's look up apples. And it appears on our list. Let's look up bananas. There it is on our list. As you can see, it updates every time you enter something. So this list just gonna keep building with the top five things that were most recently searched by a user. Now let's see the ways to delete. So one of them are you swipe. You can either check it or delete it. Right. And once you check it, as you can see we checked apples and it appears right here. You use the screen for the report. So let's say the apples cost us $2, and we bought five of them. Right. And then we add it and it appears right there on our total. And this is if we want to like change the theme of our app. We go over here and there's light mode, there's dark mode, so you can change this. You can also change the accent color, which is this, so make it red. Purple. Light mode it's up for user customization. Alright, we can go back and then we go to our settings page, which contains all our information. I can change my phone number for example to something else. You can change your e-mail or username. Update our profile and it just saves. It can also change our password right here. It has been updated, then if we want to log out, you can just simply log out. Boom and that is our application.

5.0 Test Plan

Testing for the LinkList grocery app was done throughout the development of the application. For example, whenever a new module was introduced, whether a major change or a simple change, the application was tested to make sure it worked well and did not break the app.

Flutter makes testing apps extremely easy with a feature called Hot Reload and Hot Restart. With these features, whenever a change is made the changes can be tested by simply saving on the development environment. This will cause the app to quickly rebuild with the new changes and demonstrate them almost instantaneously. When working through a bug that stops the application using the hot restart tool rebuilds the entire application. These two tools are really helpful in helping test changes as soon as these are made and not have to compile everything from the ground up, which takes significantly longer considering the massive amount of files needed. With all of this in mind, testing was made extremely easy for us.

Separately, we also needed to test each other's changes to the application. To do this, we used the version control tool GitHub to keep the code up to date with the latest version. However, this is where we would run into more trouble as often something that worked with one of the team members did not work on the other. To solve these problems, we needed to collaborate to find any issues.

For cloud storage on this project, we used FireBase. To test reading and writing we did it either one of two ways: pulling from the database to make sure everything connected as it should have and printing information to the screen. Another was to simply sort through the documents in the database to ensure the correct information was written to the database.

6.0 Test Report

Cart:

- Difficulty getting the correct group from the home screen.
    - Fix: Pass in necessary information into the cart screen to locate the correct document in the database.
- List overflows outside of screen.
    - Fix: Wrap the list in a scrollable widget.
- Trouble connecting the cart to the database.
    - Fix: taking a snapshot of the specific document in FireBase.
- When nothing was returned from search, a blank widget would be created.
    - Fix: Ensure anything returned from a search was not null.

Color Customization:

- Some items were initialized before a connection to FireBase was established.
    - Fix: Ensure connection to database is established before trying to work with items.
- Trouble fully implementing color system
    - Fix: Combine database storage with hardware storage to form a constant theme
- Trouble making theme constant when app was reloaded
    - Fix: Write the user's theme to storage to be pulled when app is closed

Invite System:

- Trouble implementing Invite system
    - Fix: Add a new collection to the database
- Trouble implementing username and group name into invite
    - Fix: Initiate invite object with current user's information

7.0 Conclusion

In conclusion, the development process for this application is a dynamic and user-centric path marked by agility, collaboration, and continuous room for improvement. The process follows an iterative and adaptive approach. The team prioritizes transparency, flexibility, and regular feedback from users. Our use of version control systems such as Git and meticulous testing ensures integrity of the code and application robustness.The emphasis on design thinking principles and seamless information flow contributes to the creation of this user-friendly and versatile grocery application. This aligns with our commitment to delivering a stronger and constantly evolving solution for modern grocery shopping needs.

Appendix A - SRS

1. Introduction

This grocery list application is designed to improve the efficiency of a person's day to day experience while shopping. User's will be able to create, manage and execute their personal grocery list to help make their shopping experience more convenient. There will also be many new features such as synchronization between multiple devices, sorting and categorizing store items, scanning of barcodes, predictive text, recipe integration, and even tracking of the user's budget! This application strives to enhance the way a user can plan, shop, and budget their groceries to help make their shopping experience the best.

1.1 Goals and Objectives

The main goals of this application are to:

1. **Streamline Grocery Shopping**- simplifying the creation and management of user grocery lists, making it more user-friendly and efficient. Users will be able to quickly add items, categorize, and organize their list for the best shopping experience
2. **Strengthen Shopping Efficiency**- we are aiming to strengthen shopping efficiency by providing many features such as predicting text input, which will help users save time and errors as they shop
3. **Improving Budget Management**- this application will also help users in managing their grocery budget. This application will offer a report to assist users with budgeting .

2. Overall Description

Our grocery list application will be a user-centric software solution created to increase the efficiency and convenience of managing, creating, and executing grocery lists for users. The application aims to make the grocery shopping experience easier for users by offering a range of features, which include multiple device synchronization, sorting, predictive text, recipe integration, and many more features. This application strives to better the way users plan and budget their groceries in order to improve their overall shopping satisfaction.

2.1 Product Environment

The application operates in a dynamic and evolving digital world. It is mainly designed to be used on mobile devices such as smartphones and tablets using operating systems such as iOS and Android to make sure it reaches as much accessibility for everyone. It will also integrate with cloud-based services to allow users to sync their grocery lists across different devices with no data problems

3. Usability Requirements

Usability requirements are a must have for this application. The application will have a visually appealing user interface with easy navigation and straightforward decisions with the grocery lists to help accommodate everyone, even people with disabilities. The data entry side of the system must also be prioritized. Users will be able to add, edit, and remove their grocery items from their lists effortlessly through multiple ways; barcode scanning, manually changing the list, or voice recognition some time in the future. We will also make the application inclusive for accessibility standards such as keyboard navigation and screen reading. Users will also get clear error messages with guidance to resolve these issues. Included with this is confirmation messages whenever someone adds an item or for notifications. There must also be a user guidance program or a tutorial to help users understand how the application works.

3.1 Functional Requirements

R.1.0 Launch Page

- R 1.1 Login button redirects to login page
- R 1.2 Register button redirect to create account page

R 2.0 Login Page

- R 2.1 Login with username
- R 2.1 Password
- R 2.3 Recover password (optional)

R 3.0 Register - Create Account – Page

- R 3.1 Email
- R 3.2 Password
- R 3.3 Username
- R 3.4 First Name
- R 3.5 Last Name
- R 3.6 Phone

R 4.0 Settings Page

- R 4.1 Update email
    - R 4.1.2 Requires confirmation

R 4.2 Update password

R 4.3 Update Phone Number

R 4.3 Change Names

<u>R 5.0 Create a Group</u>

R 5.1 Group Name
R 5.2 Each group will have a grocery cart

<u>R 6.0 Leave a Group</u>

R 6.1 Adds option on home screen to leave Group

<u>R 7.0 Invite Others to Group</u>

R 7.1 Send another user a "Group Invite" using user's email
R 7.2 User can either decline invite or accept within 30 days
R 7.3 When accepted, new member will be added to the group

<u>R 8.0 Adding Items to Cart</u>

R 8.1 Any user in a group will be able to add an item to the cart
R 8.2 A user can search for an item in a search bar or add it by name
R 8.3 Added items will appear in everyone's Cart menu

<u>R 9.0 Removing Items from Cart</u>

R 9.1 Users can remove items from cart one of two ways:
- R 9.1.1 Deleting an item entirely from the cart
- R 9.1.2 "Purchasing" item signifying item is bought

<u>R 9.0 Generated Reports</u>

R 9.1 Generate an expense report for users

3.2 Safety Requirements

Because we are dealing with sensitive personal information, it is important to keep this information safe. This is why in Section 3.1, we have implemented certain features which we will expand on here. To change an email or password, we require confirmation to make sure the user is the one changing the information. In another instance, we require "invites" into groups so a user cant be added to an unwanted group.

3.3 Documentation and Training

The plan for training will be through an easily understandable UI to assist users with understanding the app's functionality. Documentation will be done both inside the project's code and within other files. These include the Ganntt chart that we use for our requirements that we plan to use to track the progress.

3.4 User Interface

The UI should have a productive feel to it while still maintaining ease of use. Since the app is designed for anyone from the age of 13 and above, the app should be family-friendly. The features should be familiar such that 90% of users should be able easily navigate through it and use the requirements without much hassle.

4. Usability

Our grocery app aims to simplify shopping through an intuitive and seamless user experience. The interface will be designed with the user in mind, making it easy to navigate and access key features. Users can add items to their cart manually. Features will be organized and sortable for quick access, while reports provide helpful budget oversight. Sharing features will be simple to use, enabling coordination between users. The focus throughout is on an intuitive, user-friendly experience that simplifies and streamlines shopping.

Appendix B - SDD

1. Introduction

Welcome to our system design document for our Grocery Application. This will be a new digital solution for people to manage, plan out, and make better decisions for their grocery experience. In this fast paced world we live in right now, convenience and efficiency are what sets applications apart from each other. With an application such as our grocery application, it builds a connection between the digital world and our everyday life. The application will offer many intuitive features such as categorization of store items, budget reports, and many more! This document goes into explanation into the design and architecture complications that make up our application. As you explore this document, you'll gain better insight into how our application uses the modern day technologies to provide the best grocery shopping experience for users ranging from any backgrounds.

1.1 System Overview

The grocery application is a comprehensive software application created to better the grocery shopping experience for all types of users with different backgrounds and preferences. The system works as a hub for creating, managing, and optimizing each grocery list. Users can use manual entry through the keyboard to add items to their grocery lists. The system will employ categorization and sorting methods to present seamless organization based on the user's preference. Users who are conscious of their spending habits will benefit a lot from the budget reports.

2. Design Considerations

2.1 Design Assumptions

With the evolution of technologies over the years and the many more to come we must be aware of assumptions to be made for our application. If a new operating system becomes significant in this volatile market it can be problematic for our product. The most used operating systems as of now in the mobile application industry are iOS and Android. We would have to undergo major updates if a new operating system drops. Another design assumption we must be aware of is the possible changes of functionality. As the application evolves through time, features and functionalities we had during our initial design could become outdated. Also, we must keep up to date with functionalities that are of value with external services. Our recipe database and price comparison features could lead to issues if we don't constantly update them to keep up with newer trends.

2.2 General Constraints

Some constraints we must be mindful of are as follows:

1. *Hardware limitations:* the application could have some issues being ported onto different devices from different years. Each device can have its own restrictions that it operates on. Careful optimization can eliminate this issue
2. *Data privacy*: the application can have issues on how the user data is stored and collected for the application. We must follow HIPAA guidelines to best collect data in a legal way
3. *Internet connectivity*: Internet speeds around the world can vary due to many different factors
4. *Accessibility:* We have to ensure that the application is accessible for all types of users. Many people with disabilities have a limited way of using applications so this will impose constraints on the development process to have options for them as well

2.3 Goals and Guidelines

1. *Simplicity*: The biggest goal is to simplify the grocery shopping experience. Prioritizing user-friendliness ensures that all of the features are efficient and intuitive so it can be easy to use regardless of a user's level of proficiency in technology
2. *Data security*: This is an important point to protect the privacy and user data. Users must have trust in our application to share their shopping data to ensure the best experience
3. *Continuous Improvement*: We must seek user feedback to try and keep the application improving as time passes. We will prioritize regular updates to keep the application relevant and up to date with industry trends

3. Architectural Strategies

When conceptualizing our grocery list application's architecture, a variety of key decisions and strategies were put in place to guarantee the app's scalability, efficiency, and top-tier user experience.

3.1 Programming Language and Development Framework

- **Decision**: Utilize Flutter with the Dart programming language for both Android and iOS platforms.
- **Reasoning**: Flutter offers a cross-platform solution, allowing for faster development and maintenance with a single codebase. Dart, as Flutter's language, offers strong support, efficient performance, and a rich standard library that simplifies many tasks.

3.2 Reuse of Existing Components

- **Decision**: Integrate Flutter's set of packages for features like cloud synchronization.
- **Reasoning**: By tapping into Flutter's ecosystem we can benefit from well-maintained and widely-used packages ensuring faster development.

3.3 Extension Plans

- **Decision**: Design the app in an integrated fashion with the aid of Flutter's widget-based architecture.
- **Reasoning**: This approach allows us to scale or modify features with ease catering to ever-evolving user demands.

3.4 Error Detection and Recovery

- **Decision**: Incorporate thorough logging and error reporting mechanisms specific to Flutter's ecosystem.
- **Reasoning**: Early detection and rectification of issues will foster user trust and satisfaction.

3.5 Memory Management

- **Decision**: Utilize Dart's garbage collection capabilities and follow Flutter best practices to manage memory efficiently.
- **Reasoning**: Proper memory management ensures smooth performance across a myriad of devices including older ones.

Armed with the power of Flutter and these architectural strategies, we are confident in delivering a state-of-the-art application that caters to the modern user's needs and expectations.

4. System Architecture

The grocery list application is architected with a primary focus on usability, scalability, and modular design. The architecture is aimed at striking a balance between a highly responsive front-end and a robust back-end, while ensuring that components are loosely coupled for better maintainability.

4.1 Major Responsibilities and Roles

The system's primary responsibilities include:

1. User management: Account creation, login, authentication, and data synchronization.
2. List management: CRUD operations on grocery items, categorization, and list sharing.
3. Budget management: Production of expense reports
4. Collaboration: Sharing lists, real-time updates, and multi-device synchronization.

4.2 System Decomposition

Given the nature of our application, we've decomposed the system into three major components:

1. **Client Layer**: Developed using Flutter, this layer is responsible for UI/UX, user input processing and presenting data.

2. **Application Logic Layer**: This layer acts as the bridge between the client and data layers. It houses business logic, request processing, data transformation, and acts as the primary point for authentication and authorization.
3. **Data Layer**: Stores user profiles, grocery lists, and handles the bulk of the application's data operations. In addition, it manages data synchronization across devices.

4.3 Component Collaboration

The Client Layer communicates with the Application Logic Layer to fetch, process, or send data. This interaction is achieved through API endpoints, and data is typically transmitted in JSON format. The Middleware, in turn, interacts with the Data Layer to carry out CRUD operations and ensures data consistency across the board.

4.4 Rationale Behind Decomposition

We chose this particular decomposition for a few key reasons:

- **Scalability**: This design allows us to scale each component separately, depending on the demand. For instance, if we experience a surge in users, we can scale our backend resources without affecting the front-end.
- **Performance**: With separate layers handling different tasks, we can optimize performance at each level, ensuring faster data retrieval, processing, and presentation.

4.5 Existing System Components

Since we're building on top of pre-existing platforms, it's essential to mention that these platforms come with their pre-built components and libraries. However, in our architecture, we've only delved into the custom components we're developing. In essence, our architectural strategies are aimed at creating a sustainable, efficient, and user-friendly ecosystem ensuring our grocery app stands out both in terms of functionality and user experience.

4.6 Data Requirements

Date Required for Running Application

- The application will require a way to get the grocery items that users will be able to add to their cart.
- In order to use the application, users will need to create an account, entering account information such as their email, first and last name, phone number, as well as a username.

4.6.1 Data Model

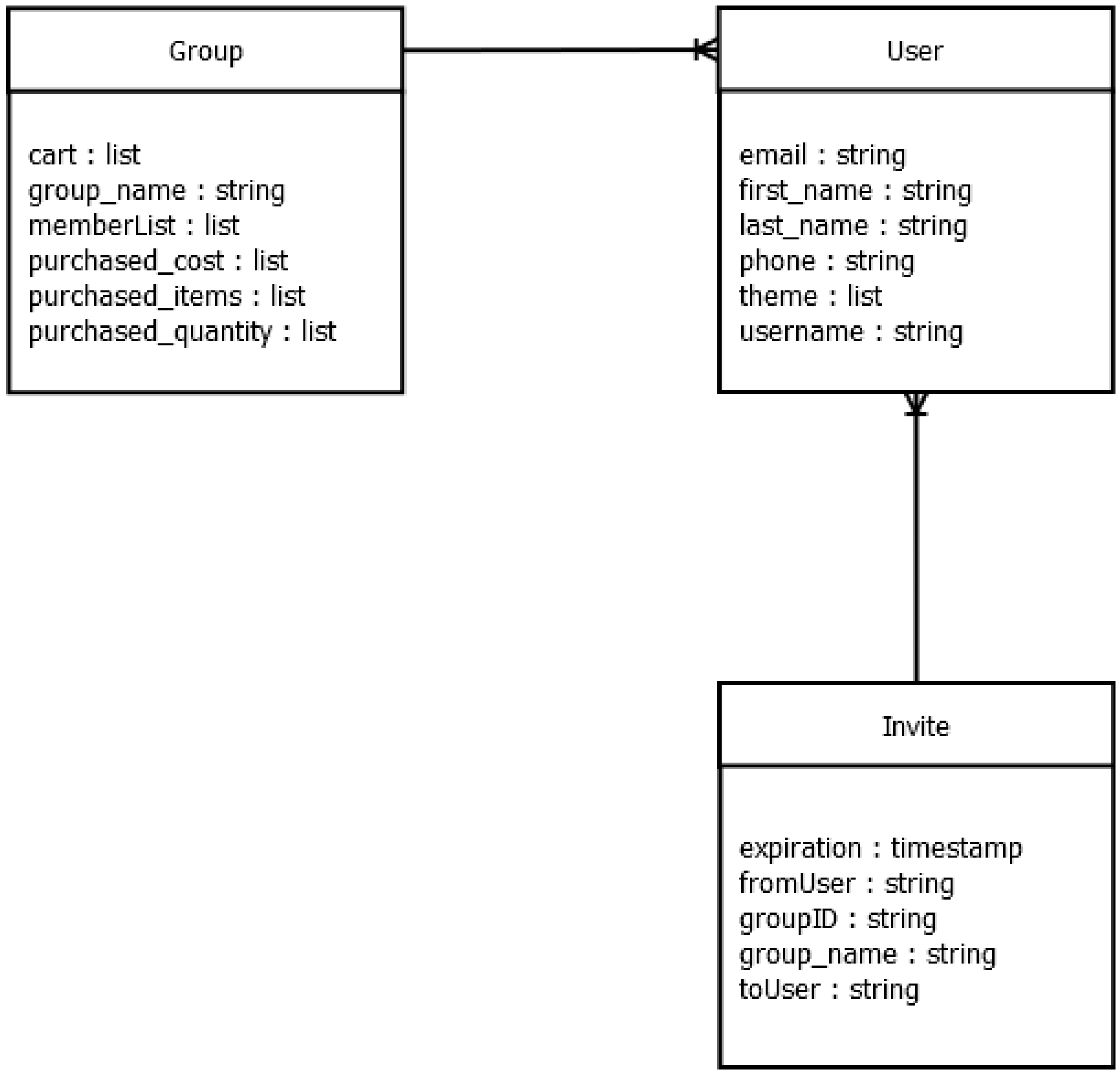


Figure 4: Data model diagram

4.6.2 Data Dictionary

User

- Email--String
- First Name--String
- Last Name--String
- Phone-String
- Username--String
- Theme--List

Group

- GroupName--String
- MemberList–List

- Cart–List
- PurchasedCost–List
- PurchasedItems–List
- PurchasedQuantity–List

Invite

- Expiration–TimeStamp
- fromUser-String
- groupID–String
- groupName–String
- toUser–String

4.6.3 Communications Interfaces

The app will communicate via a central cloud-based server called FireBase. The app uses the collections and documents within FireBase to integrate synchronous communication between users and to store user data.

5. Policies and Tactics

5.1 Code Compilation

For this project, Flutter will be used. Flutter compiles code natively and therefore loses little in productivity on devices.

5.2 Programming Guidelines

As with any other project involving multiple people, in order to maintain consistency there must be guidelines for everyone to adhere to. Since this is a software project, this will concern naming conventions for files or variables. The following guidelines should be followed:

1. Each variable, method, and file name should be descriptive (ex. int name)
2. Use of camel case (ex. int firstVariable)
3. Classes should be entirely capitalized (ex. class GroceryCart)

Separately, documentation is important. It is not necessary to include in-line comments to every method or line in the code. Rather the author of a file should take careful consideration as to what needs detailed descriptions and what only needs a general description. Example in pseudocode:

```
//This method takes two integers, adds them together, and //returns the result
public int addTwoVars (int firstInt, int secondInt){
        return firstInt + secondInt //returns the sum
}
```

5.3 Files for the Project

This project will require design, styling and logical forms in order to function. Furthermore, we will need FireBase to handle all data and communication.

5.4 Organization of Files

Because of the scope, we can assume there will be a lot of files in this project. Therefore, rather than having all the files thrown together in one place, the project will have folders of related material. For example, all the UI code will be maintained in one place, all the functional code in another. An example of the organization is below:

UI Form Design:
- LogInPage
- HomePage
- SettingsPage

UI Controller Sheets:
- LogInControl
- HomePageControl
- SettingsControl

Databases
- Groups
- Users
- Invites

By using this method, it becomes easy to find related items by keeping all of them in one place.

5.5 Requirement Traceability

In order to keep track of progress on the project, we will use our Ganntt Chart. While there exists software for progress, our current chart is excellent for all of us to constantly check the status of our requirements.

Our plan is as follows:
1. Assign someone a task
2. The person will provide a weekly update in the **Complete %**, and **Memo** sections of the chart.
3. When completed, the person will provide completed status on the chart.

5.6 Testing and Maintaining the Software

In order to mitigate risks, testing will be done constantly. This can be done through either debugging, or full white/black box tests. For example, as we begin working on a new major requirement, we can test to make sure data input and output works in each module. When the requirement is done, we can test that single requirement in its entirety by compiling the application.

5.7 Final Deliverable

The finalized version of the app will be turned in as a compiled library with documentation inside, as well as the current version being updated on the GitHub organization

6. Detailed System Design

6.1 Dart

Classification
Language

Definition
Dart is an object-oriented programming language developed by Google and is widely used for mobile, desktop, and web-based applications. This will be the main language used within our application.

Responsibility
This component is responsible for the backbone of our project and interfacing with APIs and other users.

Constraints
There are a few constraints of this language. The language is still new and is currently under construction, and the amount of internet resources for the language are limited.

Composition
This component contains classes, functions, and data types used within the program.

Interactions
This component is used to make the framework for our project, so it will interact with every other aspect within our scope. This language is mainly used by our framework, Flutter.

Resources
Dart's garbage collection capabilities will help manage memory efficiency. This language will also be in charge of local and cloud storage..

Processing
This language will handle all processes and calculations within our app.

6.2 Flutter

Classification
Framework

Definition
Flutter is an open-sourced framework used to build applications. It utilizes the Dart language to build natively compiled, multi-platform applications. This will be the main backbone of our application, allowing the users to access our app on both IOS and Android app stores.

<u>Responsibility</u>
This component is responsible for being the backbone of the application and designing the UI.

<u>Constraints</u>
The main constraint for this component is that it is still a new framework, and the Android/IOS updates may be a bit behind, so the framework may not be fully updated.

<u>Composition</u>
This component will contain the entire app UI and any background user tasks

<u>Interactions</u>
As this is the backbone of the app, it will interact with almost all components.

<u>Resources</u>
This component will use system memory and storage and use the Dart language.

<u>Processing</u>
This component will process all the user interactions and GUI processes.

6.3 FireBase

<u>Classification</u>
Database

<u>Definition</u>
FireBase is a cloud-hosted database that allows developers to store and synchronize data in real-time. It is comprehensive with both mobile and web applications.

<u>Responsibility</u>
This component is responsible for storing user data in the cloud and providing a synchronous user experience across all devices.

<u>Constraints</u>
The main constraint for this component is the learning curve with how to properly handle data requests and storage in a clean, efficient manner.

<u>Composition</u>
This component will contain all user data and interactions.

<u>Interactions</u>
This component will provide synchronous user interactions.

<u>Resources</u>
This component will use cloud storage and require an active network connection.

Processing
This component will process all the user interaction with the stored data.

6.4 Client Layer

Classification
Layer
Definition
This layer is responsible for the UI, user input, and presenting the outputs from the app. This is the layer that the user will interact with and view. It will be following UI guidelines such as *Apple's Human Interface Guidelines* and others in order to present an user-friendly experience that is accessible to all users.

Responsibility
This component will allow the user to interact with the application. It will communicate via the second layer to the Data Layer. It will handle any UI/UX functions and containers, as well as any user inputs.

Constraints
This component needs a very low percentage of issues, because any problem will directly have an effect on the user's experience.

Composition
This layer will contain the Flutter framework and contain containers, such as UI text and buttons, as well as functions that communicate and take in user input.

Interactions
This component will directly interact with the user and communicate with/via the Application Logic layer, all while using the Flutter component as a building block.

Resources
This component will pull memory resources from the system and use the Flutter framework.

Processing
This component will process all user interactions and the communications into the Data layer.

6.5 Application Logic Layer

Classification
Layer

Definition

This layer bridges the other two layers together. It handles certain logics and request processing, and will be the point for user and account authorization. This will be what provides the user with an efficient, well processed application.

Responsibility

This layer is responsible for some minor processing and also manages the communication between the Data Layer and the Client Layer. It is also responsible for managing data transformations and authentication.

Constraints

If there are any mistakes or throttle points in the communication between the three layers, it could seriously hinder the efficiency of the program and decrease user satisfaction.

Composition

This layer will comprise communication channels, data transformation functions, authentication functions, and some minor algorithms.

Interactions

This component will interact and be the bridge between the two other layers as well as a channel for the authentication, providing an efficient experience for the user.

Resources

The other two layers will use this component as a communication channel, and the app will use this component for minor calculations and data transformation.

Processing

This component will deal with data transformations and communication between the two other layers.

6.6 Data Layer

Classification

Layer

Definition

This layer will store the user’s profiles, grocery lists, and any user connection with other application clients. This layer will handle the bulk of the application’s processing and manage the data synchronization across devices.

Responsibility

This layer will handle the bulk of the application’s processing and manage the data synchronization across devices. It will also communicate with any APIs and send it to the other layers.

Constraints

The constraints at this layer are the limits of the programming itself and the limits of the APIs used.

Composition
This layer will comprise algorithms, API usage, and layer communications.

Interactions
This layer will interact with the system storage and memory, as well as the Application Logic Layer.

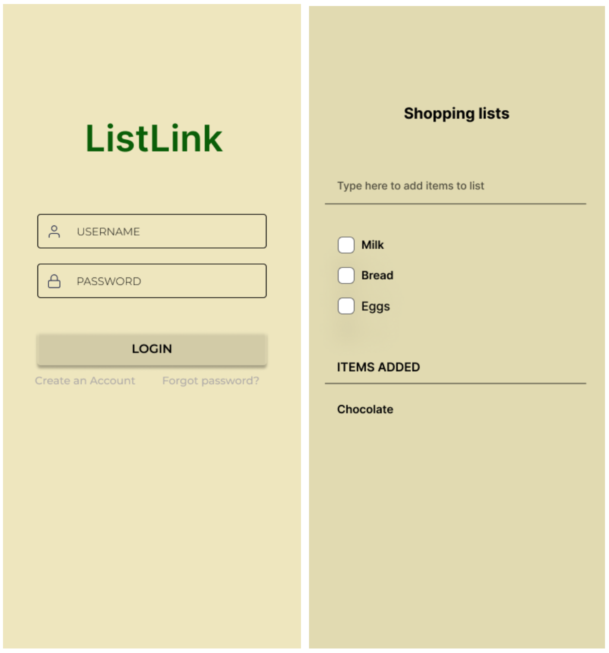
Resources
This component will be used by the app to do the majority of its processing. It will also deal with the storage and memory for the app.

Processing
This component will process all storage functions, API communications, and app processes.

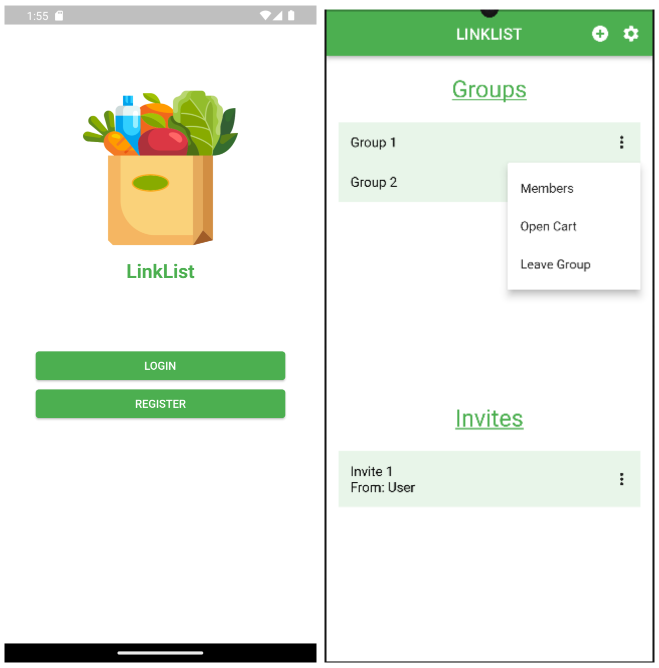
Appendix C - Design Concepts

The following are the original and current design concepts.

<u>Original UI Design Concept (Login and Home screens)</u>



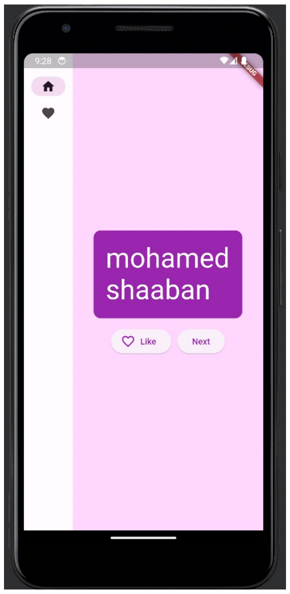
<u>Current UI Design Concept (Login and Home screens)</u>

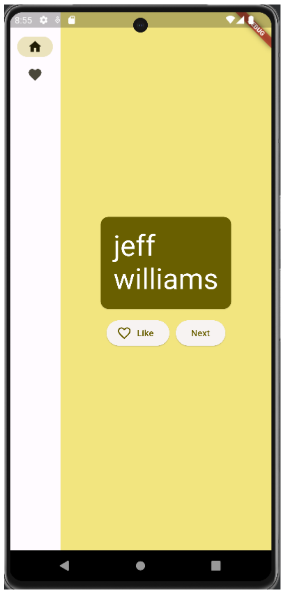


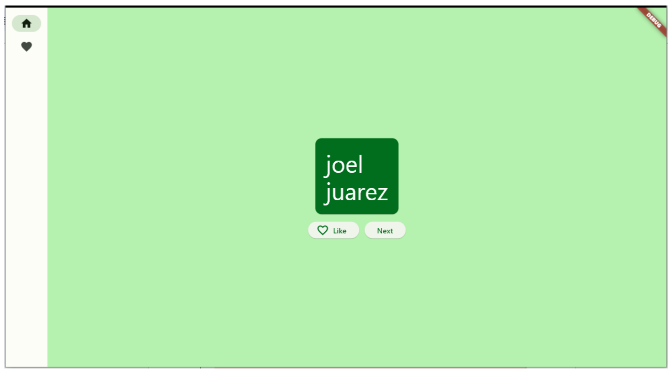
Appendix D - Flutter Tutorial Results

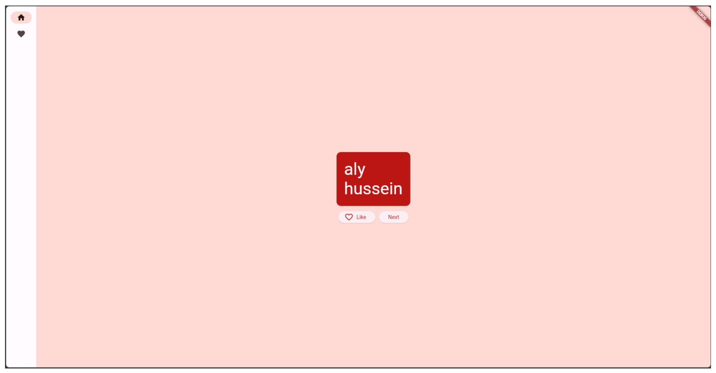
The following are each of our own edited results of the given Flutter tutorial for requested proof of completion. Each iteration is on a different device type listed below:

- Mohamed Shaaban - iOS
- Aly Hussein - Edge
- Joel Juarez - Windows
- Jeff Williams - Android


mohamed shaaban
Like
Next


jeff williams
Like
Next


joel juarez
Like
Next


aly hussein
Like
Next