SP-5 Grocery Application - Red

System Design Document

Version 2.0

11/23/2023

Prepared by:

Aly Hussein	Mohamed Shaaban
Jeff Williams	Joel Juarez

Table of Contents

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 As the application evolves through time, features and functionalities we had during our initial design could become outdated.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

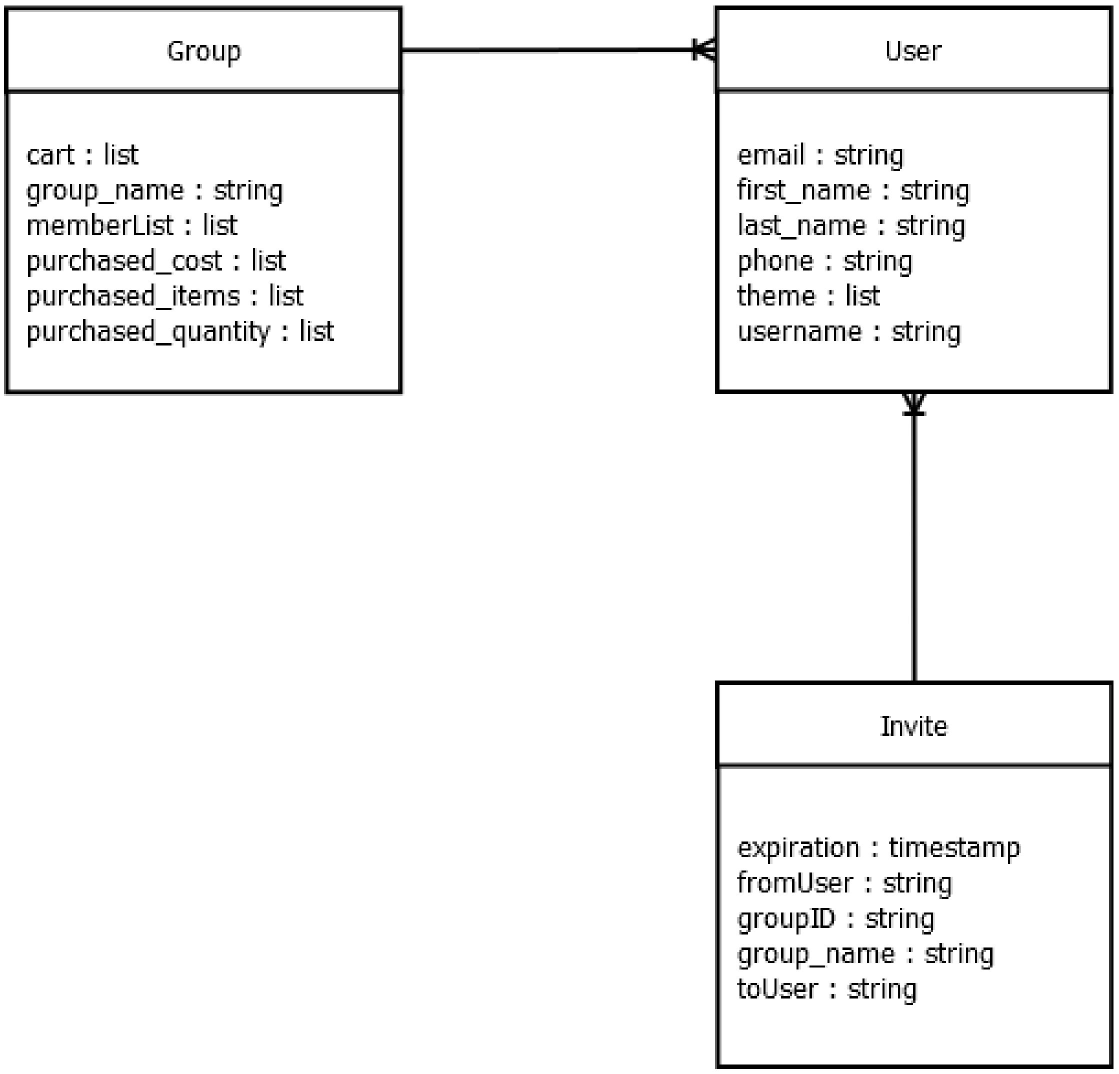
Since we're building on top of pre-existing platforms, it's essential to mention that these platforms come with their pre-built components and libraries. However, in our architecture, we've only delved into the custom components we're developing. In essence, our architectural strategies are aimed at 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

<u>Date Required for Running Application</u>

- 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



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.

Responsibility
This component is responsible for being the backbone of the application and designing the UI.

Constraints
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.

Composition
This component will contain the entire app UI and any background user tasks

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

Resources
This component will use system memory and storage and use the Dart language.

Processing
This component will process all the user interactions and GUI processes.

6.3 FireBase

Classification
Database

Definition
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.

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

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

Composition
This component will contain all user data and interactions.

Interactions
This component will provide synchronous user interactions.

Resources
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.

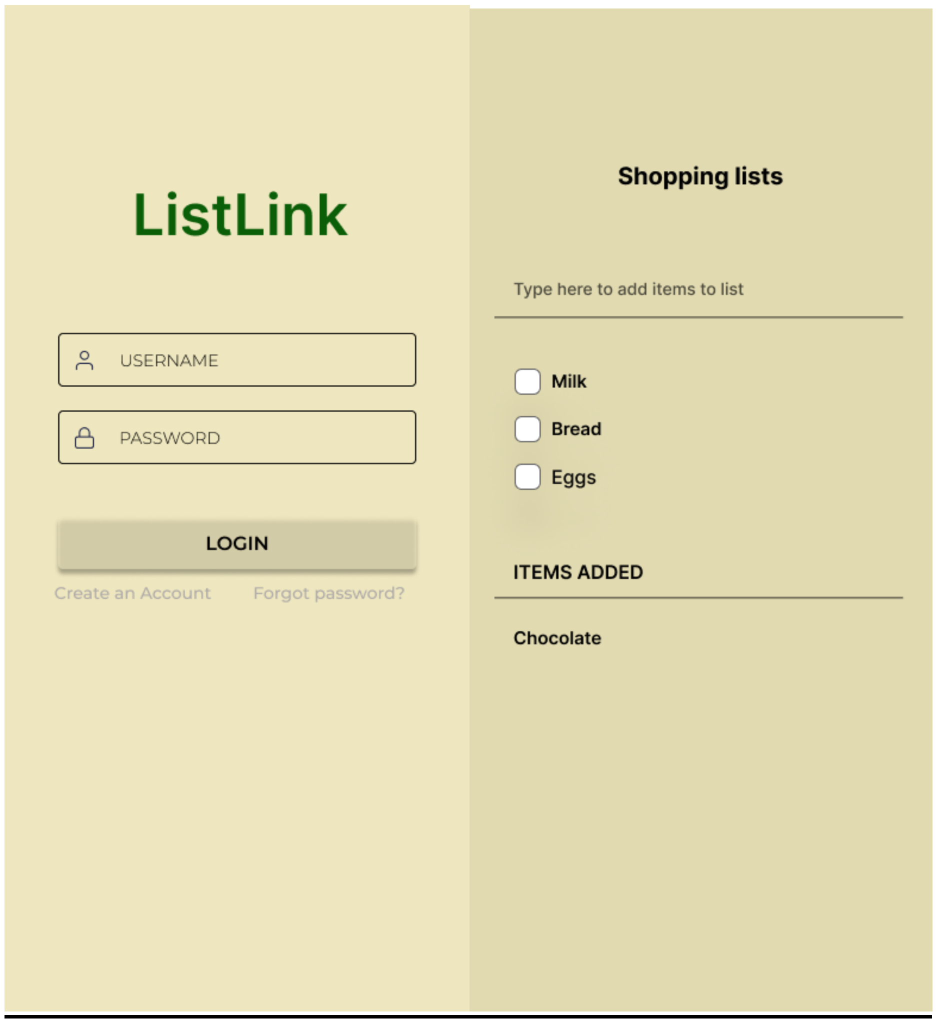
Interactions
This layer will interact with the system storage and memory, as well as the Application Logic Layer.

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.

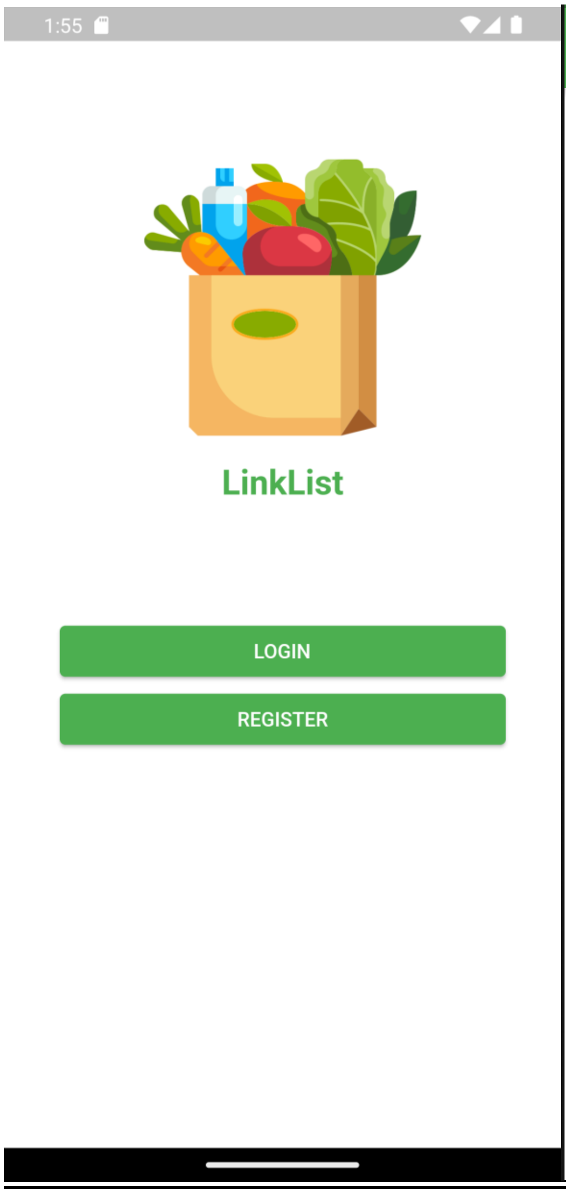
Processing
This component will process all storage functions, API communications, and app processes.

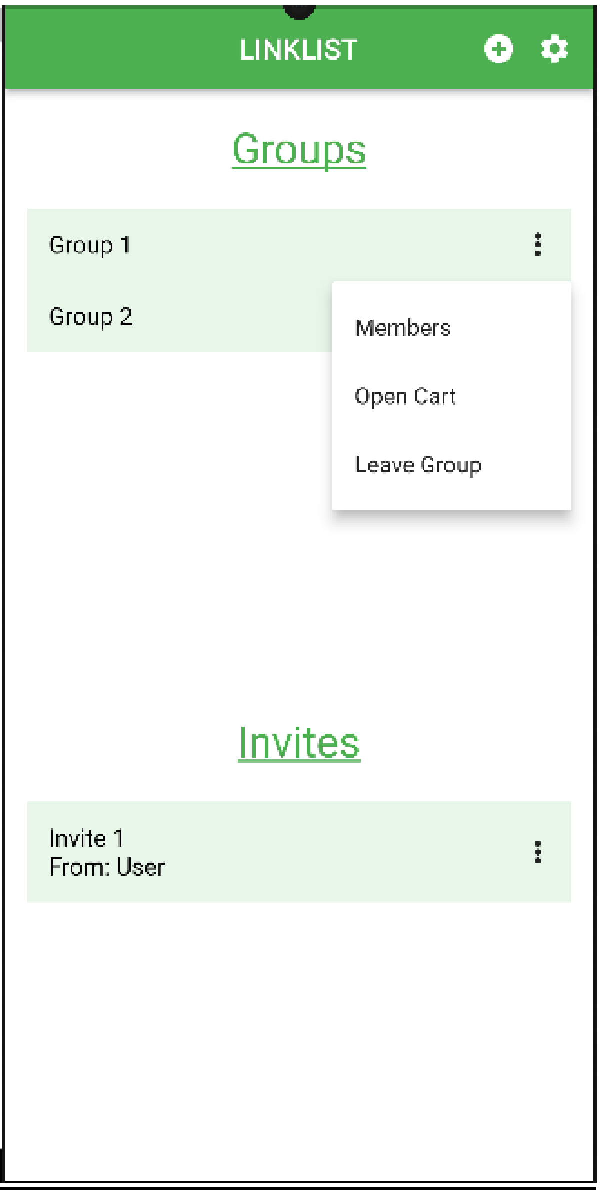
Appendix

<u>Original UI Design Concept</u>



Current UI Design Concept


1:55
LinkList
LOGIN
REGISTER


LINKLIST
Groups
Group 1
Group 2
Members
Open Cart
Leave Group
Invites
Invite 1
From: User