Final Report

Dynamic Grocery List

CS 4850-02

Spring 2023

Professor Perry

04/07/2023

SP5-Yellow-Dynamic Grocery List

By: Phillip Magnicheri, Scott McCandless, Kevin Galdamez, Ashia Hawkins

Table of Contents

Abstract

The goal of the Dynamic Grocery List application is to ease the process of grocery shopping for roommates, families, friend groups, or individuals. The core objective of the project will be to develop a mobile application to allow users to create a shared grocery list. To accomplish this, the application should grant users the capabilities to view or edit the list, and to save the lists both locally as well as in a database.

Our team developed the Dynamic Grocery List using the React framework while the associated database was developed using Microsoft Azure. The primary, or phase one, objectives during the development of the project were: list design, user login, database design, website design, and final app design/mockups. Once phase one was completed, secondary , or phase two,objectives of the application will be to explore monetization options, searching for coupons, and price comparison.

Requirements

Project requirements were established through several preliminary team meetings. During these meetings we identified the primary goals we wanted the

Dynamic Grocery List to accomplish, specific characteristics end users should have, and anticipated assumptions and constraints we would be operating under. After about two weeks of discussion, we were able to finalize a list of requirements which are summarized below:

Project Goals

- Develop an app that allows for the base functions of creating, editing, sharing, saving, and deleting a grocery list.
- Develop a database to store created lists and information regarding connected users.
- Design a well designed user interface which efficiently carries out the user's actions.

Requirements Summary

1. Login
1.1 A user must be able to login to the application using their username and password.
- Each user must have a unique ID associated with this login.
1.2 The program must display created user lists upon login.
- The program must display any previously created lists or display "no lists available".
2. List Manipulation
2.1 The program must allow the user to create a list and add any desired items.
- The user must be able to create/name lists and add up to 50 items.
2.2 The user must be able to delete items from the list as well as delete the list itself.
- Items should be deleted immediately while lists should be moved to a trash bin.
2.3 The user must be able to share their list to other users.
2.4 Users must be able to designate roles for shared lists (i.e viewer/editor).

- Viewers and editors will have distinct permissions to take action on a shared list.
2.5 After creation, lists must be saved both locally and in the database.
2.6 Users must be able to search for a certain list as well as items in a list.
- The program should display results in order of relevance according to input.
2.7 Users may enable notifications for actions performed on a list.
- I.e. user should be notified when a list has been shared/unshared with them
3. Data Integrity/Security
3.1 The users' information must be encrypted.
- The user's personal information must be secured to prevent outside access.
3.2 Access to the program database must be protected and recorded.
- Access should only be granted to specific individuals, and any actions taken by said individuals must be kept in a log.
3.3 The program database must backup data at least daily.
- Backups will maintain a screenshot of the database which can be used to recover lost or corrupted information.
3.4 The integrity of the data should be maintained by the database.
- Data integrity will be maintained by handling synchronous actions to a shared list. 3.5 The size of the database must grow in accordance with the number of users.
3.6 The user's device should contain local storage of at least 500 MB.
3.7 The user's allocated space on the database must adhere to the project's policies.

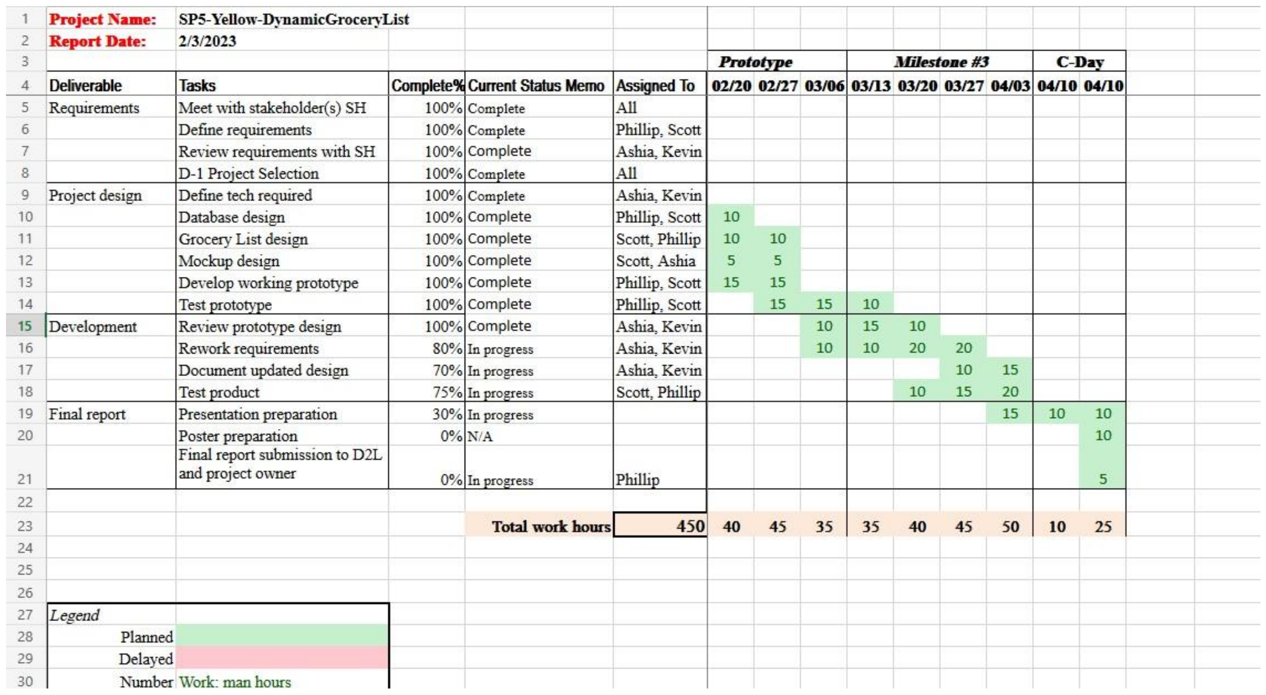
3.8 The user interface should be clear, consistent, and responsive.
- The interface must be designed so that it is simple and intuitive to users
3.9 The user interface should disguise the inner workings of the program.
3.10 The program should be compatible with the latest versions iOS and Android Operating Systems.
3.11 Under normal operation, each given page must load within 2 seconds.
- The program should not have excessive load times that hinder user experience
3.12 After the app crashes, the app must load in between within 10 seconds.
- Extra time should be allowed for the program to reload/resync data
3.13 The server hosting the database should be able to carry the load of all active users.
- The server database should be able to allocate/deallocate space as needed

Project Management and Planning

In order to ensure that the project was progressing according to schedule, the team made use of several different tools.The primary tool that was the foundation of our planning was the Project Plan and Gantt chart which was developed early in phase one.The project plan was essentially the ‘skeleton” for the project and almost every aspect of the project was built using one or more of the initial designations found here. The Gantt chart was especially useful in allowing team members to see upcoming deadlines as well as determine how much time should be allocated to which specific task each week.

To coordinate team meetings as well as other communications the team used a Discord server. The primary team meetings were held weekly every Monday and Wednesday from 11:00 -12:15. Additionally, there was an optional team meeting every Friday at the same time which was used to either go over any deliverables due and to check in before the weekend. Secondary means of communication included emails, primarily used to communicate with our project advisor, as well as phone numbers, for emergencies.

Code management and version control was handled through the use of Github. This allowed for shared code updates and replication of code as well as allowing other members to peer review code.Work for this project was conducted via pull requests from said repository. Pull requests were required to undergo peer review from at least one team member before it was allowed to be merged with the primary code base in order to minimize merging conflicts.



Project Name: SP5-Yellow-DynamicGroceryList
Report Date: 2/3/2023

					Prototype			*Milestone #3*				*C-Day*	
Deliverable	**Tasks**	**Complete%**	**Current Status Memo**	**Assigned To**	**02/20**	**02/27**	**03/06**	**03/13**	**03/20**	**03/27**	**04/03**	**04/10**	**04/10**
Requirements	Meet with stakeholder(s) SH	100%	Complete	All									
	Define requirements	100%	Complete	Phillip, Scott									
	Review requirements with SH	100%	Complete	Ashia, Kevin									
	D-1 Project Selection	100%	Complete	All									
Project design	Define tech required	100%	Complete	Ashia, Kevin									
	Database design	100%	Complete	Phillip, Scott	10								
	Grocery List design	100%	Complete	Scott, Phillip	10	10							
	Mockup design	100%	Complete	Scott, Ashia	5	5							
	Develop working prototype	100%	Complete	Phillip, Scott	15	15							
	Test prototype	100%	Complete	Phillip, Scott		15	15	10					
Development	Review prototype design	100%	Complete	Ashia, Kevin			10	15	10				
	Rework requirements	80%	In progress	Ashia, Kevin			10	10	20	20			
	Document updated design	70%	In progress	Ashia, Kevin						10	15		
	Test product	75%	In progress	Scott, Phillip					10	15	20		
Final report	Presentation preparation	30%	In progress								15	10	10
	Poster preparation	0%	N/A										10
	Final report submission to D2L and project owner	0%	In progress	Phillip									5
			Total work hours	450	40	45	35	35	40	45	50	10	25

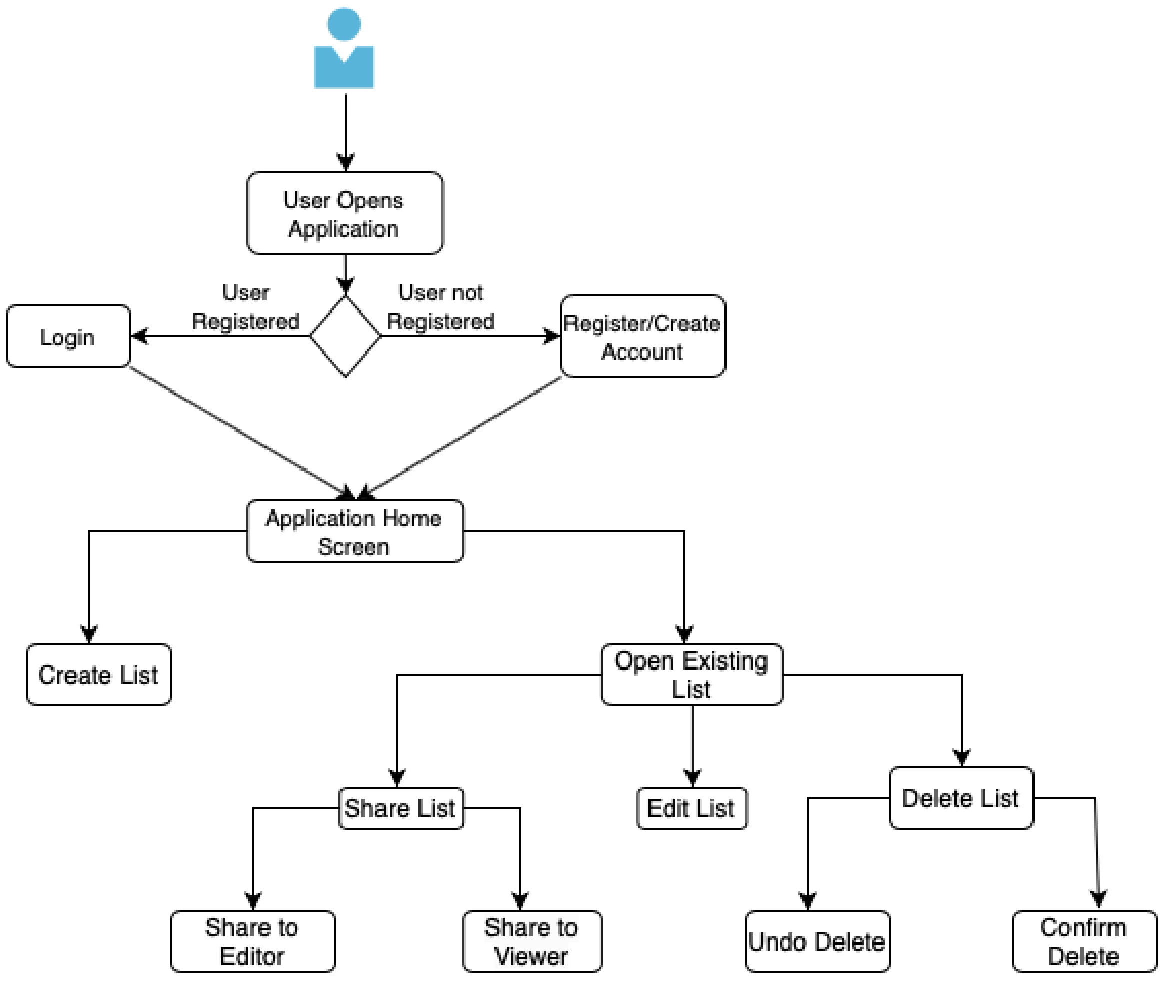
Legend	
Planned	🟩
Delayed	🟥
Number	Work: man hours

Figure 1: Team Gantt Chart

Design

Application Architecture

The mobile application was organized into four primary components: list creation, list editing, list sharing and list deletion. To begin, a user logs in to the application with their credentials or has the option to create an account. Once an account is created the user either views the lists they have already created or can create a new list. When a list is opened the user will then have access to edit the list and add or delete any items, share a list with another user and designate them as a viewer or editor, or the user can delete a list along with secondary options to undo a deletion or confirm a deletion. A figure summarizing a high-level abstraction of these interactions is included below.



Database Architecture

The database implementation was done in MySQL using Microsoft Azure's database hosting service. Azure provided services to allow us to handle the integrity of user credentials as well as provided the ability to allow the database to scale according to the amount of space needed to store user data.

The first collection of attributes created in the database were designed to hold all of a user's account and login information. These attributes are as follows:

A. Username
B. Password
C. First Name
D. Last Name
E. Email

Once a user enters data for these attributes not only are they able to log into the application but they also create the email attribute that will be used as a reference when needed.

The next collection of attributes are used in the creation of a standard, non-shared note. For all non shared lists, the variable that tracks whether or not a list is shared, Shared_Status, is set to false. The variables used in this collection are:

A. User_ID
B. Note
C. Note_ID
D. Date
E. Note_List

The User_ID is a numeric ID created by the database which is an identifier that serves as a reference that establishes a user as the owner of a note. For example,, any time the database is queried to show all notes belonging to user 2023 , the database would query all notes with a User_ID that is equivalent to 2023. The note attribute represents the given name of a list. Note_ID is a numeric identifier created by the database that is used to find requested notes; this is done to ensure the correct note is queried in the event that a user creates lists with identical names.Lastly, the date identifier represents the date the note was created and the Note_List variable stores all of a users notes via a relationship with a given User_ID .

Deletion of a note shares the same attributes used in the creation of a note. When a user chooses to delete a note; the database will query the Note_ID of the selected note in order for the note to be either confirmed for deletion or for the deletion to be canceled. Upon confirmation, the Note_ID will be deleted and removed from the notes associated with a given user via User_ID.

Editing of a note also makes use of both the Note_ID and User_ID attributes. A user selects a note that they wish to edit, and the database queries the Note_ID associated with the user's User_ID and open's the desired note. The user is then free to make any necessary edits.

Given that the user desires to create a shared note, there are two options available. The user can either create a fresh shared note or the user can convert a standard note into a shared note.

Creation of a new shared list allows a user to create lists that are viewable and/or editable by other users. The attributes used to establish such a list are:

A. Owner_ID
B. Share_ID
C. Edit_Rights
D. Viewer_Rights

The Owner_ID attribute is used to designate the user who owns the shared list and therefore has all associated privileges such as role designation and editing rights. Owners are able to search for users they wish to share a list with by inputting the email of the user they wish to find. Emails are used instead of usernames in order to ensure uniqueness. Share_ID is used similar to the Note_ID attribute as it references the requested shared note. The Edit_Rights and Viewer_Rights are boolean attributes to determine in what manner a user who has a list shared interacts with said list. Viewer_Rights is enabled by default; Edit_Rights, however, is determined by the list owner.

If a user desires to convert a standard note into a shared note, the User_ID and Note_ID are used. When provided with this information, the note matching the given Note_ID has its Shared_Status updated to true, and is added to the Note_List of the designated User_ID.

Mockups

Application mockups were developed using the defined requirements. The tool used to design all mockups was Marvel, a free online design software that is popular in web and web-based design. The design of the application was divided into three sections: Login View, Home View , and List View. Login view displays a welcome screen, text boxes for username and password and the login button. Upon login, the home view is used to display all of a user's lists with an icon to indicate whether the list is standard or shared. Finally the list view is displayed once a user selects a list from the home screen and allows the user to manipulate the list as they see fit, with buttons for editing, sharing and deletion.

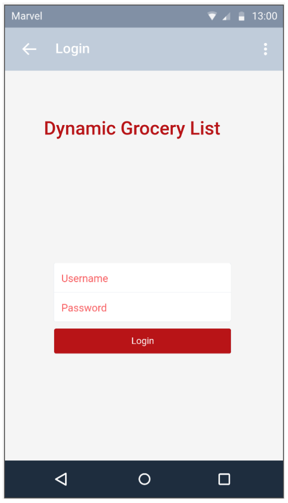
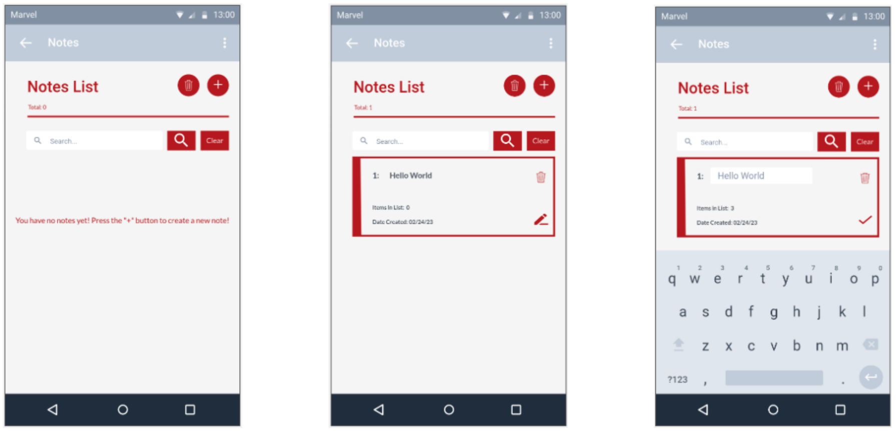


Figure1: Marvel Login Screen



You have no notes yet! Press the "+" button to create a new note!

1: Hello World

Items in List: 0

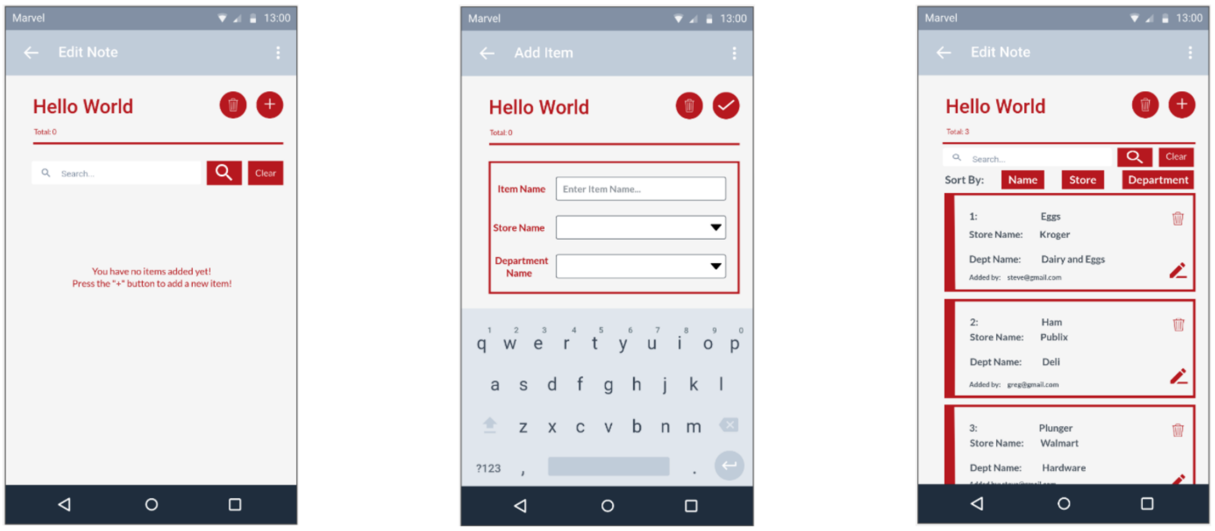
Date Created: 02/24/23

1: Hello World

Items in List: 3

Date Created: 02/24/23

Figure 2. Marvel Home Screen Mockups



You have no items added yet!
Press the "+" button to add a new item!

1: Eggs
Store Name: Kroger
Dept Name: Dairy and Eggs
Added by: steve@gmail.com

2: Ham
Store Name: Publix
Dept Name: Deli
Added by: greg@gmail.com

3: Plunger
Store Name: Walmart
Dept Name: Hardware

Figure 3. Marvel List View Mockups

Development

Development began with research into the Flutter and React frameworks in order to determine which would be most suitable for our application. After each team member did both research and simple tutorials, we decided that React Native was more appropriate for this project. Afterwards, we then spent time looking at similar list-building applications to analyze the design choices made by other developers. This provided a lot of insight into the requirements and attributes which needed to be addressed in the beginning stages of project development. Additionally, this process aided in discovering how we should approach connecting the front and back ends.

Further development of the project was divided into three major sections: (1) List Functionality, (2) Database Functionality, and (3) UI Refinement. The first two sections were developed in parallel in order to ensure the front and back ends were properly linked and able to communicate back and forth. Afterwards, the majority of the team's time and resources were spent in analyzing the UI and formulating ways it could be improved or simplified. Improvement focused on quality of life changes to the user, such as performance enhancements, responsiveness/feedback, and learnability.

The development of sections one and two, while time consuming, were not terribly complicated.There were several resources at our disposal for the team to reference whenever we ran into a dead end or encountered a complex problem. Such resources included W3schools for common coding language knowledge and syntax ,React for environment specifications, Youtube for react set up and knowledge, and finally Microsoft's azure website for cloud based server information . The language of choice for the front end was Javascript, naturally as React is Javascript based; the backend was hosted on Microsoft Azure, which takes advantage of SQL Server capabilities.

Section one development began with the task of simply creating a list and then displaying that list's contents in the application. Originally, the application displayed both the list name and several of the first few items in the list. However, this feature was later changed so that only the list name was displayed on the home screen, and once a user selected a given list name, the elements of that list would then be queried and displayed. After list creation was accomplished, the additional functionalities of list editing and list deletion were added on top. The login page was then implemented so that we were able to add multiple lists for a single user and ensure that the basic list functionalities were performing correctly. At this time, the focus shifted to allowing customers to share lists. List sharing involves two aspects: (1) determining which user(s) you want to share a list with, and (2) designating a viewer or editor role to any shared users. In the front end, the largest decision for shared lists was implementing sharedness in a way the user would find intuitive. This included determining which actions the user would take to share a list and set permissions and determining how the list should be displayed on the screen so that a user can identify the difference between a shared list and a local list. Originally the plan was to have two different pages to house each type of note. However, in order to improve the flow of the application, the current version contains both shared and non-shared notes on the same screen. In order to distinguish the types of lists, a different icon is used for shared notes in order to notify the user.

Development of the database was closely related to the development of the actual application, therefore they were built in an iterative fashion mostly simultaneously. Node.js was the tool that was chosen in order for the front end application to connect to the database. By using Node.js instead of other alternatives, we were able to take advantage of its lightweight structure which allowed for fast response times. Additionally, this helped our application to be scalable should the opportunity for monetization arise.The database's first functionalities were allowing for a user to simply create and store a list. This involved creating variables for user identification and list identification, which allow for the desired list to be pulled from the database upon a user's request. Once completed, the next goal was implementation of list editing and list deletion. List editing was fairly straightforward, but list deletion took a little more effort to get operating as desired. We did not want a user's list to be permanently deleted after only one click of a button; as this is often something that can be seen as an annoyance in mobile applications. Therefore we decided to implement confirmation messages in the delete menu that will prompt a user to only proceed with deletion if they are absolutely sure they do not wish to use the given list anymore. This change was implemented as a direct result of discussion about how the database should handle deleted lists and whether or not we wanted to ask for confirmation or allow for a grace period before the list is deleted. Development, afterwards shifted to the login screen. This process was also relatively straightforward, the only minor holdup was concerning whether we wanted to use a user decided username or simply use the user's email. However, we decided to use a username mainly due to privacy; more user's would likely prefer to display a name that they can decide on the screen, rather than be forced to have their email displayed.

Finally, the database development's final task was implementing shared lists. This step involved additional research and discussion regarding scenarios such as race conditions, which could arise from simultaneous editing, actions on deleted lists, and distinctions between different users' items. This led to the decision to add list viewers and editors in shared lists as well the identification of the method by which users would select other users to share lists with. In this case, email address was chosen over userID as the means for finding users to share a list as email addresses are already unique and will not run into any problems of duplicate values. Role designation was added to allow users the option to let added users edit the shared list or not via the editor or viewer roles, respectively.

Once functionality of both the application and the database was at an acceptable level, development effort shifted into optimizing the user interface. The initial UI, referred to as the Alpha version, was relatively barebones and displayed minimal information on the screen; essentially only showing the necessary buttons and data that needed to be visible during the development process. This process was relatively simple as the updated, or beta, UI was built on the foundational alpha version, but added useful tips and button symbols that will allow the user to intuitively navigate throughout the application.

The overall development process of the application modeled that of a real-world project, and it required a substantial amount of time, research, and planning. There was only one major problem that occurred in the development process, which was a result of an update to the Node.js library. About halfway into the development process, the update was automatically installed, which caused the database to not correctly communicate with the front end. Luckily, it only took a few hours to completely resolve, but given that it occurred right around the same time that the list sharing functionality was being implemented, it caused some unanticipated delays. This development process left us with the Beta version of the app which encompasses the primary objectives (phase 1 goals) discussed at the start of the project as well as a foundation for any expansions or monetization efforts.

Testing

Test Plan

The test plan for the mobile application was created using the requirements previously listed in this report. After testing was performed on the application the results indicate that the application is ready for publication. Tests results were recorded on a pass or fail basis and any tests that were not able to be performed are marked with a not applicable or N/A. Details of tests performed can be found below:

Test Reports

Section	Test	Pass/Fail
1.Login	Login page can create new account with userID and password	PASS
	A user must be able to login to the application using their username and password.	PASS
	The program must display created user lists upon login.	PASS
	Login page must authenticate existing users	PASS
	Login page must have account recovery options	N/A
2. List Manipulation	The program must allow the user to create a list and add any desired items.	PASS
	The user must be able to delete items from the list as well as delete the list itself.	PASS

	The user must be able to share their list to other users.	PASS
	Users must be able to designate roles for shared lists (i.e viewer/editor).	PASS
	After creation, lists must be saved both locally and in the database.	PASS
	Users must be able to search for a certain list as well as items in a list.	PASS
	Users may enable notifications for actions performed on a list.	PASS
3. Data Integrity/Security	The users’ information must be encrypted.	PASS
	Access to the program database must be protected and recorded.	PASS
	The program database must backup data at least daily.	PASS
	The integrity of the data should be maintained by the database.	PASS
	The size of the database must grow in accordance with the number of users.	PASS
	The user’s device should contain local storage of at least 500 MB.	PASS
	The user’s allocated space on the database must adhere to the project’s policies.	PASS

	The user interface should be clear, consistent, and responsive.	PASS
	The user interface should disguise the inner workings of the program.	PASS
	The program should be compatible with the latest versions iOS and Android Operating Systems.	PASS
	Under normal operation, each given page must load within 2 seconds.	PASS
	After the app crashes, the app must load in between within 10 seconds.	PASS
	The server hosting the database should be able to carry the load of all active users.	PASS

Conclusion

Upon finishing the completion of the last deliverables, there are a couple of takeaways from this project. One of the primary takeaways was gaining first hand experience in how influential project management skills and software engineering concepts are in development. A larger than anticipated portion of time was spent in planning design and listing requirements while, outside of one or two instances, the actual writing of the code was fairly straightforward.

However, the time spent in the initial planning stages undoubtedly played a role, as less upfront consideration would surely have resulted in more problems during development. Additionally, an appreciation for the amount of effort spent in software development was gained. Between project management, code development and review, mockup design and review, requirement eliciting, and many other tasks; it is evident how crucial it is to have software developers that are well rounded and flexible in order to adapt to the situation they are in.