Design Document Ver. 1.1

February 8, 2018

Team Name:
Gnosis Solutions

Team Members:
Kalen Wood-Wardlow, Christopher Simcox, Thomas Back,
Kristoffer Schindele

Project Sponsor:
Dr. Leverington

Faculty Mentor:
Dr. Leverington

Table of Contents:

Introduction…………………………………………………………………………………2
Implementation Overview……………………………………………………………….....3
Architectural Overview……………………………………………………………………….4
Module and Interface Descriptions………………………………………………………7
Implementation Plan……………………………………………………………………..10
Conclusion…………………………………………………………………………………...12

Introduction

Grocery shopping chains in the past few decades have offered rewards programs that allow customers to follow daily deals and shop wisely, while the store collects data about customer shopping habits. Unfortunately, there are rarely any special programs that allow the customer to benefit from collected data such as: what certain items are in stock and helpful details about those items like seasonal availability. With the exception of curbside pickup or delivery programs, there are rarely any tools for the customer that will allow them to plan and shop more effectively. For this reason, customers develop an affinity for a “familiar” store that they visit exclusively to cut down on time spent looking for items or sales. This is done without any consideration that there may be stores available in closer proximity to their location or needs.

A multitude of organizational list applications exist on the iOS and Google app stores. Some of these are specifically catered towards grocery store items, such as the app Out Of Milk. This app allows a user to create a virtual list that can carry details about each item entered that are more customizable than just a written list. These item details can include price, quantity, category, and additional information such as sales tax. As the shopper collects items, they can check each one off and reduce the number of items displayed in the list to streamline their shopping experience. They will then be able to save certain lists for future reference, modification, and sharing with other users. In the end the app’s main purpose is to serve as a handy organizational interface to make sure all required items were bought, packed along with a few extras features. One functional oversight found in these apps is that they contain information about items that customers are accustomed to finding at their regular store. If the app could be modified to include the store it was found at, and its location within the store, as well as share such information with other users via a central, customers could use this data to shop more efficiently.

Crowdsourcing applications rely on community participation to enter accurate information about a subject contribute to a larger network of information that allow all users to benefit in ways that cannot be achieved otherwise. Gnosis Solution’s vision is to create a mobile application that leverages shared information, and an intuitive user interface to provide benefits to grocery shoppers. Some examples of other applications that use this concept include:

- Waze, a GPS navigation app that lets users enter real time traffic information, such as accidents, construction, and traffic checks.
- Google Places and Yelp, which notice when a user is located at a particular business and asks them to write reviews and take pictures to effectively provide other consumers with firsthand experience with a business.

Similarly the application Gnosis Solutions is developing will incentivize users to reap mutual benefits through the following features:

- An attractive list making feature that allows users to quickly and efficiently organize their list information.
- A central database through which users can share the details of each item.
- A local database on the user's device that ensures they can access shared information no matter where they are.
- An intuitive user interface focused on touch-based interactions that minimizes obstruction to user interaction while maximizing efficiency.

Gnosis Solutions were brought this idea by Northern Arizona University lecturer Dr. Michael Leverington. Dr. Leverington has identified these key underutilized areas in the current market seeks to capitalize on the potential of individual user data stored in shopping list applications. Our overall goal is to make users' shopping experience less cumbersome. Our team plans to solve the issues outlined above by creating a shopping list application that relies on GPS technology and users' personal knowledge of product details at certain stores to facilitate other features, such as a shopping route generator pathfinding algorithm and price comparisons for items between stores.

Implementation Overview

As stated previously, our solution to this goal is to create a cross-platform mobile application to serve as the user's hub for accessing and creating their lists, as well as being able to compare prices of their list between stores in their surrounding area. Our approach to this centers heavily around a cloud database in which user's account and list information will be stored, as well as the shared storage for pricing on specific stores. With the storage remaining in the cloud, the application will provide the interface and User Experience for inputting the information to be shared and accessed by user's later.

The front-end mobile application will be built using the cross-platform rendering platform, React-Native. React works by building certain components that serve as tags to render UI elements. These tags can pass data, and contain methods for reacting to user input and updating the state to re-render or hold new data passed in. Data can either be passed from child to parent or upwards from parent to child. This allows us to have parent and child tags that make up UI components, with data passed throughout.

In order to utilize the Firebase database an API constructed off of the Firebase API supplied from google has been built. This allows us to access endpoints from the

database, manage and transport path information, and save and load data easier into our tags.

In order to streamline the login process and leave the user account management up to a third party, we plan to integrate login through Facebook and Google. This involves introducing a special API provided by each company.

Geolocation services will be provided via React-Native's Geolocation API. This will be utilized to handle finding the user's location, and remembering the locations of stores and markets.

On top of this, we will be using specific libraries accessible through the node package manager that add content for React-Native. So far these include a router library named "Flux" which manages routing of separate screens as well as the data that is passed in between them. Next is the React-Native-Elements library that helps for features on the list screen such as the checkboxes, and sliding to open features on the list view.

Architectural Overview

In order to continue to explain this project's solution and give an idea of what our complex and efficient software does it is important to give an architectural overview of how the system is working to deliver such a solution. The diagram as follows is the workflow architecture of the system proposed:

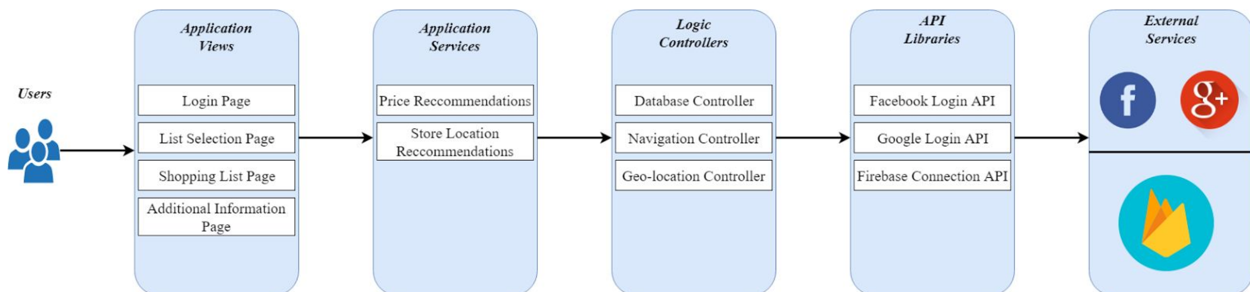


Figure 1

This system design is closely related to a model view controller architecture. Where the view is the area where the user can see content. The model is how the information is being passed and the controller handles all of the background logic to make the system do work. With the exception that this application does rely on external packages for functions ranging from network communication to general app navigation. Here is more a more in-depth explanation of the above diagram (Figure 1):

Users

In the architecture of this application it is assumed that users, zero or more, interact with the application via either the Android or IOS platforms. From this point of launching the app then the architecture's next level is the application views. The application views will go into more detail but this is how the users see an interface on their chosen devices.

Application Views

The application views portion of the application architecture describes what the user will actually see. Since the goal is to create a great experience for the user the app gives them a number of pages that will assist them in using the application. Each entry in this section in the diagram above is a separate, page, view in the application. Data is passed along between these views as necessary. Now how the user visually interacts and views content is defined it is important to go a layer deeper into the next section about some cool services offered by the architecture.

Application Services

The application services are specific actions, passive and non passive, that the app takes to enhance the user's experience. The user cannot directly view or interact with these services besides the notifications they receive as a result of these services. The two most important services to the user are the price recommendations and store location recommendations. These services would not be possible with more interfaces supporting them. This leads yet again to another step deeper into our architecture called logic controllers.

Logic Controllers

The logic controllers, developed entirely by Gnosis Solutions, support many actions in order to aid the application services and views section as needed. There are a few controllers because the app is exercising the design principle of separation of concerns. It is possible to have one large controller file but that would make things more complicated and messy. So instead the controllers are broken up into individual, more manageable, files. As of now there are three main controllers for functionality as follows:

1. Database controller that helps update, delete, add and maintain database consistency on the remote cloud Firebase database.
2. Navigation controller that assists the application in changing views from one view to another and to pass along data to be used across the entire application use.

3. Geolocation controller that assists in capturing and making geolocation data available anywhere in the application in order to assist our views and services for the user experience.

Each of these controllers are designed in a way such that they are callable when imported; similar to an API for a plug-and-play service maximized for code modularity, reusability, and efficiency. While most of the logic for the application was developed in-house, it also makes use of many remote services incorporated through external API libraries.

API Libraries

In the application there are three main API's that are used to deliver login and database functionality they are as follows:

1. Facebook login services API
2. Google login services API
3. Firebase connection API

Each of these API's are provided free of charge from the developers of the content the application is trying to access with these API's. These API's are incorporated in each of the logic controllers to cut out a lot of code and any possible error that could occur if attempted to re-engineer these API's. Finally, it is necessary to mention the external services that are used via these API's by this application's logic controllers.

External Services

As mentioned before this application uses multiple API libraries that were developed by other companies. These are used so that it is possible to easily access the following services:

- Facebook, one click and custom, login services and management.
- Google, one click and custom, login services and management.
- Firebase connection management and hosting.

Each of the above services are provided by external companies that this application does rely on to function correctly. Built into each of these are fail-safes in case the user does not have a stable internet connection at all times to still retain data. These services are managed and hosted on their own, so their stability is not a requirement of the Gnosis Solutions development team.

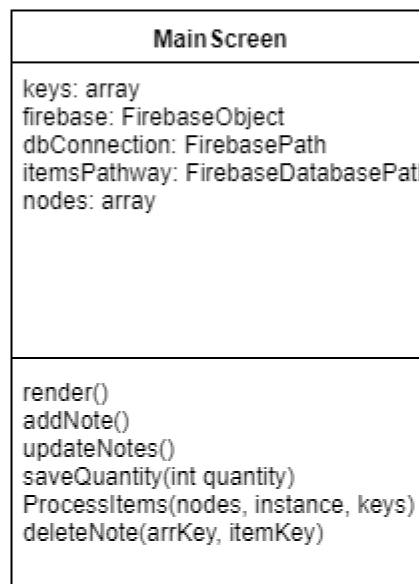
With all of this application’s architecture explained, this document now focuses on the specific technical aspects of implementation,which are detailed in the next section.

Module and Interface Descriptions

React-Native allows the definitions of components in order to separate screens, as well as components on the screen that could be complex and could benefit by separating their concerns. For this reason, the modules will consist of team-developed React components, as well as classes for database management. The first of which is the the main list screen component.

Main List Screen

This Component will serve as the template for loading in specific lists, and act as the interface for entering, editing, and deleting items.



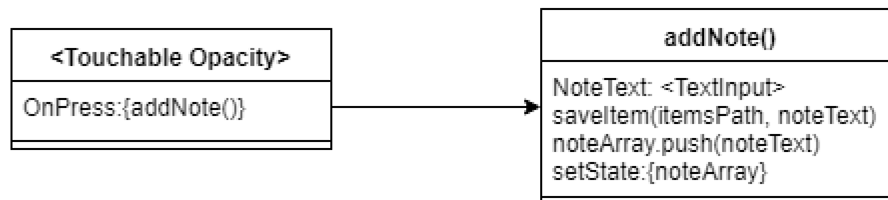
React-Native is designed in such a that the user has a default exported class act as an entire component in itself. In the main list screen, the Main Screen component serves as the top-level class for the rendering and logic.

Under the constructor there are a few constants for loading in the list data, connecting to the correct path in the firebase database, and storing these in nodes. These will be described in detail later in this section.

The keys array stores the keys of every firebase reference item that represents an item on the list. This is used to keep track of them for deletion and modification by loading them in locally. The firebase object establishes a connection with the main

remote database. The following attributes establish and save paths to certain levels of the database, for example the items pathways navigates that reference to the level where the specific items for that list would be stored, under the users' separate lists and that certain user.

Under functions, every React component must have a render function where the JXML tags will be defined for rendering and passing data through. The following diagram contains information on the components featured within the render function. After the render function the team has defined several other functions to assist in adding items, deleting items, modifying the quantity, and loading the items into the current state. First is addNote() which, when attached to the property of a UI component, will be triggered when it takes an item name entered as text input, and loads it as a new list item. Below is a diagram of how the UI component triggers the addNote().



The addNote() method will use the saveItem function defined in the database API to save the item to the item's path, and then push that item onto a noteArray. React-Native then need to re-render after updating the data held in its state object, so once the note is pushed into the array, setState must be called to render the new list item into the scrollview, making it accessible by the user in the UI.

The next function is the update notes function. This first clears the noteArray we mentioned earlier holding the list items, and reloads them with the current firebase entries, used for reloading the screen and making sure the data is accurate.

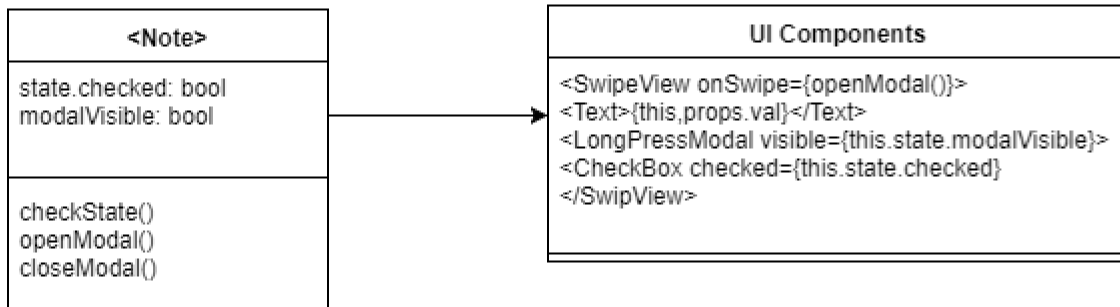
SaveQuantity() is a method that is passed down through props to another component, a modal, that will allow the editing of the quantity field of an item. This simply does it by taking the text input within that modal and updating its attribute in the database.

ProcessItems() is a method that takes in the nodes of items from the database, and loads them into the state array to be stored locally and rendered on screen.

DeleteNote() takes in the arrkey value, which holds the key for the item in the local state array, and then removes it from the array, and removes it from the database pathway, with the database key.

Note

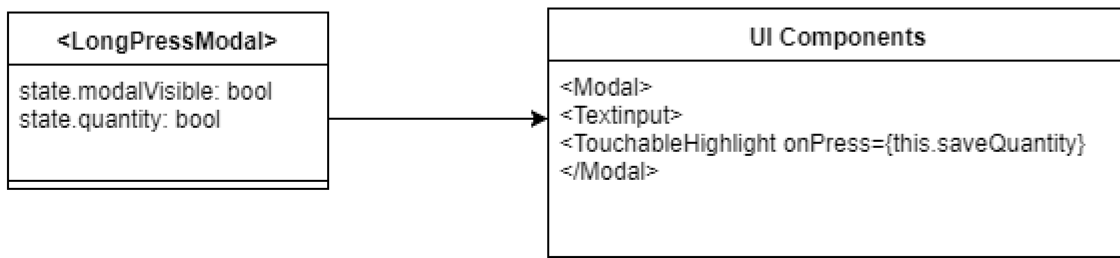
Note is the component self defined to render the list item in the scroll view. It is this component that is mapped to the note array and rendered for each item within the array. On the MainListScreen class, we have the noteArray items mapped to a note component, and we pass in that note array's text value as the list item title, this is passed into note and rendered within the note as the {this.props.val} component field.



Note's top-level UI component is the SwipeView, which is a component added from the React-Native-Elements-UI library, that enables the feature to swipe to the left or right and map a function or event to that input. In this case a modal window is opened for editing the quantity and price values for the item. To perform this, a modalVisible attribute is set in the state, which is then passed into the longPressModal component to render its visibility from the parent Note component. The checkbox that is toggled by touch is also from this UI library, and has the state.checked property mapped to it so that when it is toggled, the checkState() will update it to the reverse of whatever it is right now. LongPressModal is another self defined component that will be discussed in the following paragraph.

LongPressModal

This modal pops up above the mainListScreen in order to be a quick way to edit information about an item on your list, such as price and quantity.



Login Page

This component will be what the user first sees when s/he loads up the application. It allows the user to login to the app via facebook or google at which point s/he is assigned a userID that creates a database in firebase for the user with her/his ID as the key to accessing the database.

Login
userID: longInt firebase: FirebaseObject dbConnection: FirebasePath
render() handleLogin()

The firebase, and dbConnection are also passed into the login in order to maintain the connection to firebase.

As stated before, since every react component needs a render() function this will be used to render the login screen where the user can choose to login with google or facebook.

handleLogin() will login the user to the application by the user logging in through google or facebook and their access token given to the application which allows us to get their userID, which will then be used to access the user's database in firebase.

Implementation Plan

The plan is to have a working prototype by the 9th of March, and work for the remainder of the semester refining the user interface, performing acceptance tests, and incorporating additional features such as geolocation. Substantial progress on base functionality during the first semester of development means that most of the effort this semester can be dedicated to features and polishing. Live testing will begin at the beginning of April, and serve as the primary motivator for changes in development. The major milestones for module completion are as follows:

Functioning Prototype

Most of the back-end functionality is in place as of the writing of this document, but the user interface is yet to be fully realized. Currently, the login, main, and list pages are in place, however, additional functionality for accessing the user's settings and smooth navigation of the entire app still need to be added.

Final UI Design

While the UI navigation is still being implemented, surveys of user preferences on UI styles will be conducted to hone in on a design that provides an optimal user experience. This feedback will be used to implement different UI layouts that are distinctly different. These layouts will then be fed back to the users as a way to iteratively hone in on an optimal UI.

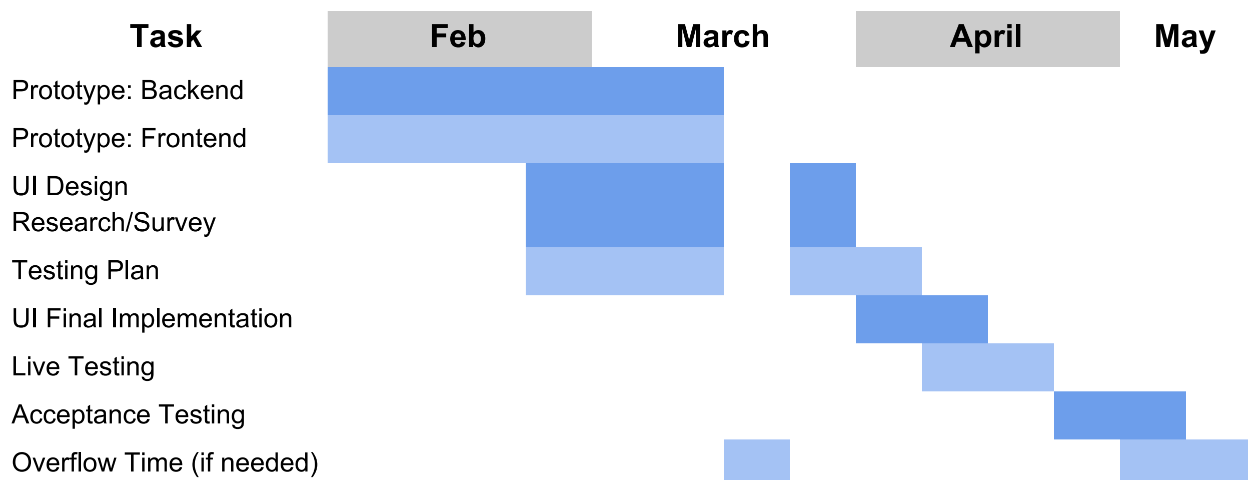
Testing

A plan to test the application will be developed from the end of February to the beginning of April; testing will begin possibly before the final plan is finished, but immediately after it is done at the latest. Similar to the UI design, the app will be released to specific user groups for trial runs over a period of approximately a month. Optimizations for UI and performance will be made during this time.

Additional Features

The final app only makes marginal use of the geolocational module, so any additional time after final UI design and testing will be dedicated to the addition of other useful features such as tighter geolocational data for store items, pathfinding algorithms for optimally navigating grocery stores, and additional uses of shared data between users.

Projected Timeline



Conclusion

Current mobile shopping list applications fail to benefit the user's shopping experience because they do not take advantage of sharing information between users. The client has requested a solution that leverages shared user information, and does so through a user interface that is 'so easy to use, they won't want to put it down'. Gnosis Solutions is satisfying these requirements with a mobile application that crowdsources information into a shared database accessible with and without an internet connection. The application also includes a streamlined and intuitive user interface that will propel it above competing software on the market. The details of this document outline the design decisions for the technologies, architecture, and classes used to build this application. The specific uses of each class, as well as a timeline for the project's development illustrate a concrete plan to bring this application from idea to implementation. Currently, the team is focused on finishing a prototype that satisfies the crowdsourcing functionality through backend software logic. However, development following the middle of March will shift to focus on the design and optimization of the User Interface, including the visible performance of the application. Live development is progressing smoothly and the next prototype will realize not only the functioning application, but a transition into the next era of shopping list applications as a whole.