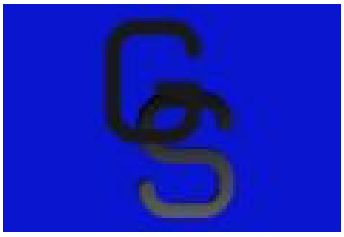
Requirements Specification



November 18, 2017

Team Name:
Gnosis Solutions

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

Project Sponsor:
Dr. Leverington

Faculty Mentor:
Dr. Leverington

Version 1.0

Accepted as baseline requirements for the project:

*Client Signature:*_____, *Date:*_____

*For the team:*_____ , *Date:*_____

Table of Contents

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. One example is the app Waze, a GPS navigation app that lets users enter real time traffic information, such as accidents, construction, and traffic checks. More examples include apps like 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.

Our application will incentivize users to enter information about grocery stores by building an attractive list making feature which will share the details of each item entered into a larger database, to be accessed by other users. Once a user prepares his list and heads to the store, in addition to checking the item off, they will be asked to

enter the location of that item within the stores layout, by simply touching the location on a map. This information can be recorded, processed, and be the source to provide future functions such as creating a shopping path for the user, or suggesting alternate stores based on comparative prices.

Gnosis Solutions was 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.

This document will delineate the functional and nonfunctional requirements we have set for completion of our application, reviewed by our team and accepted by our client.

Solution Vision:

In order to tackle the inefficiencies of modern day shopping and the lack of services provided with current shopping list apps Gnosis Solutions will build a consolidated source of information about local grocers accessible to the application users. This data will be used to provide the services such as: price comparisons, list sharing, and item location in store. This requires a database of product information, which includes the: grocery store the product was found in, location of the product within that store, price of the product, and seasons the product is normally available. A mobile application will serve as the vehicle for collecting, analyzing, and redistributing this information. Our application will assist in the user's shopping experience in two parts: list creation and preparation before heading to the store, and the shopping process while at the store. To accommodate the first phase the application must allow the user to build an electronically stored shopping list that is easy to edit, save, and share, as well as pull information from the collective database. Our list user interface will employ the following features:

- Suggesting autofill for commonly bought items on entry
- Option to add details about item, such as quantity or category
- Adding and removing items from the list
- Indication that an item has been collected
- Saving lists to user accounts
- Sharing saved lists with other users
- Editing item information at any time

- UI that guides user for rapid entry information (Suggested fillables, categories)
- UI that is conducive to rapid information retrieval

Once the user has their shopping list built and has arrived at the store, they will need to check collected items off their list. Users will be prompted at this step to enter details about their items to contribute to the collective database. Features that contribute to data entry include:

- An interface where a blueprint-esque top down layout of the store will be presented, where the user can drop a pin visually on where the item is located in the store.
    - If item's location is already entered, begin with pin on last location, user can move pin if location has changed
- A form to enter the price of the item
    - Price will autofill if price is already entered, will give the user option to update if price has changed
- Geolocation API will record the coordinates of the store the user is in, and will associate the item details with that particular store.

The user will ideally enter the details about an item onto their list which will then be synced with the records of that store's product in our remote database. Once sufficient data has been collected about the products, we can provide helpful services based on what we already know about the store:

- Based on the prices other users have entered, a price comparison interface will show at what store the user's list would be cheapest by using price data contained for each store
- Once the user has chosen a store to shop at, a pathfinding algorithm will construct a visual path of the best order to collect the items and present this visually with the same top down layout of the store used in the location entry

Figure 1 illustrates how these services will work together:

Solution Workflow Diagram

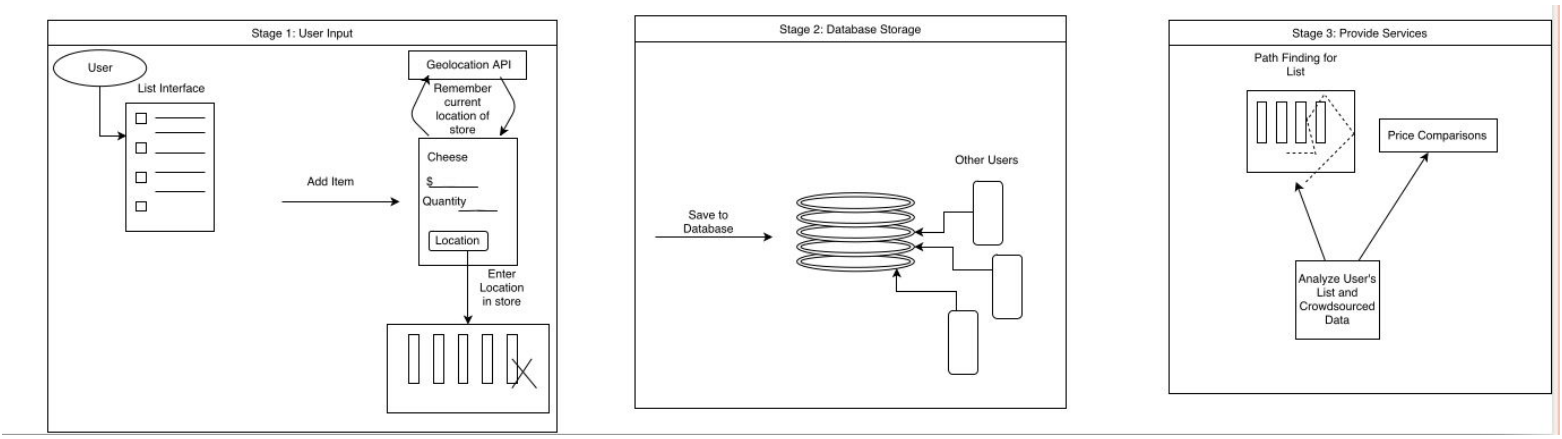


Figure 1.

These are services and features that could optimize shopping for the user by cutting down on time spent searching the store, and giving the user facility to compare prices for a list they have already created. It will create confidence that they are shopping at the best store for their interest, and will cut down on the amount of research required to shop at an unfamiliar location

Crowdsourcing the information is the best option for gathering and consolidating product and store information for users. Safeway has an API for locating products and prices, but they differ from state to state and as stated earlier Safeway is one of many grocery store chains. These factors make crowdsourcing information the most reliable method to build a database of product and store information.

As of now, this information will be used to implement two services, but further analysis could be applied to the crowdsourced data to provide additional features. The solution Gnosis Solution plans to employ is focused on empowering the user's shopping experience through information, however, crowdsourced information could also have untapped potential for research outside the current problem area.

Project Requirements

This section delineates the requirements meant to encapsulate the features that defines the full functionality of the mobile application solution. The problem domain requires a multi-platform mobile application that features an intuitive UI that allows rapid entry of list items, secure storage of item information entered by users, and

communication between a remote database and local user data. The following user requirements will describe what our system will do.

User Requirements:

UR1: Must be able to use the system like a grocery list.

For this requirement this system needs to offer users a way to input shopping list items into a grocery list. This means adding, deleting and modifying information about the products in the list such as quantity, name, store name and price. With store name and price being optional until they get to the store and check off the item on the list.

UR2: Data must be secure and reliable in transfer.

This requirement offers security to all users so that no person can view any other person's cart or data. The data on items and store's should only be used for easy entry and price suggestions.

UR3: Notify users on lower price depending on store proximity for those who opt in.

This requirement gives users a notification on lowest prices depending on stores they are close to. This is to afford users the opportunity to get the lowest price on items in their cart and also the option to not have this done to not annoy any user.

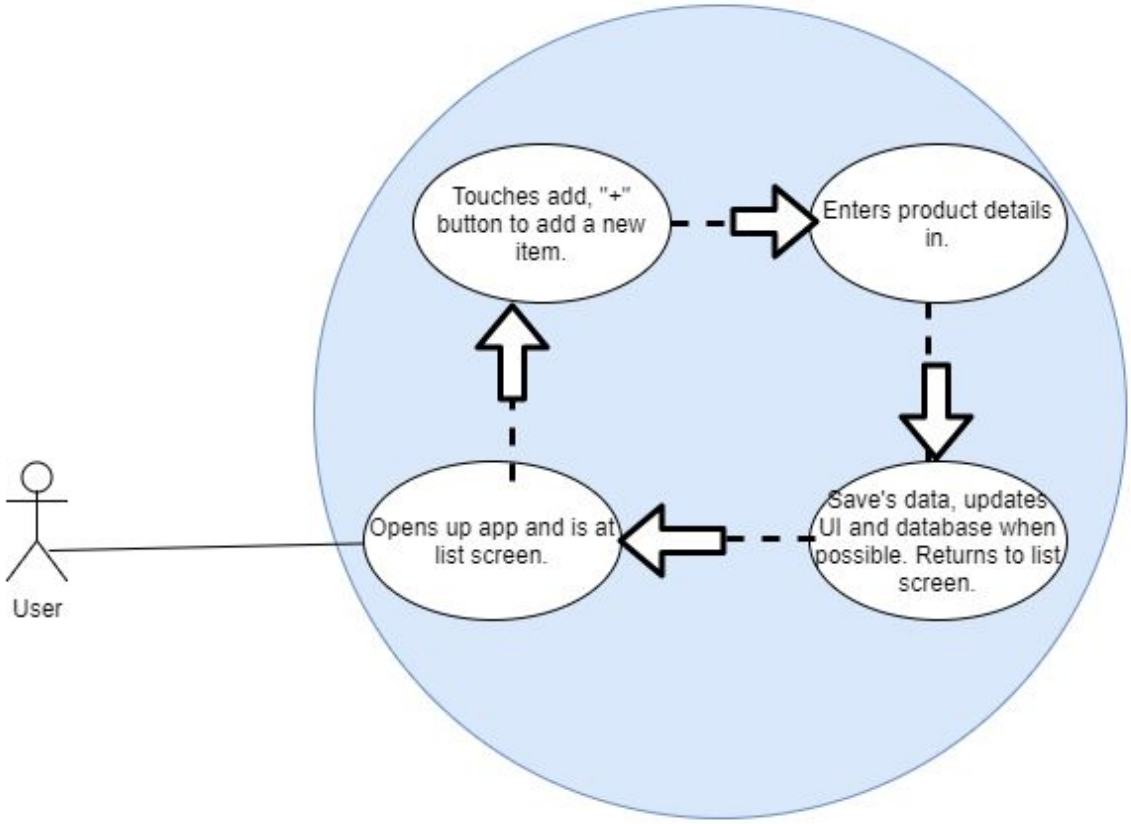
UR4: Store user data in a cloud sourced environment.

This requirement covers the client's request to have data be updated in real time and crowdsourced. The data that the user's provide must be stored in an accessible place for other user's to take advantage of.

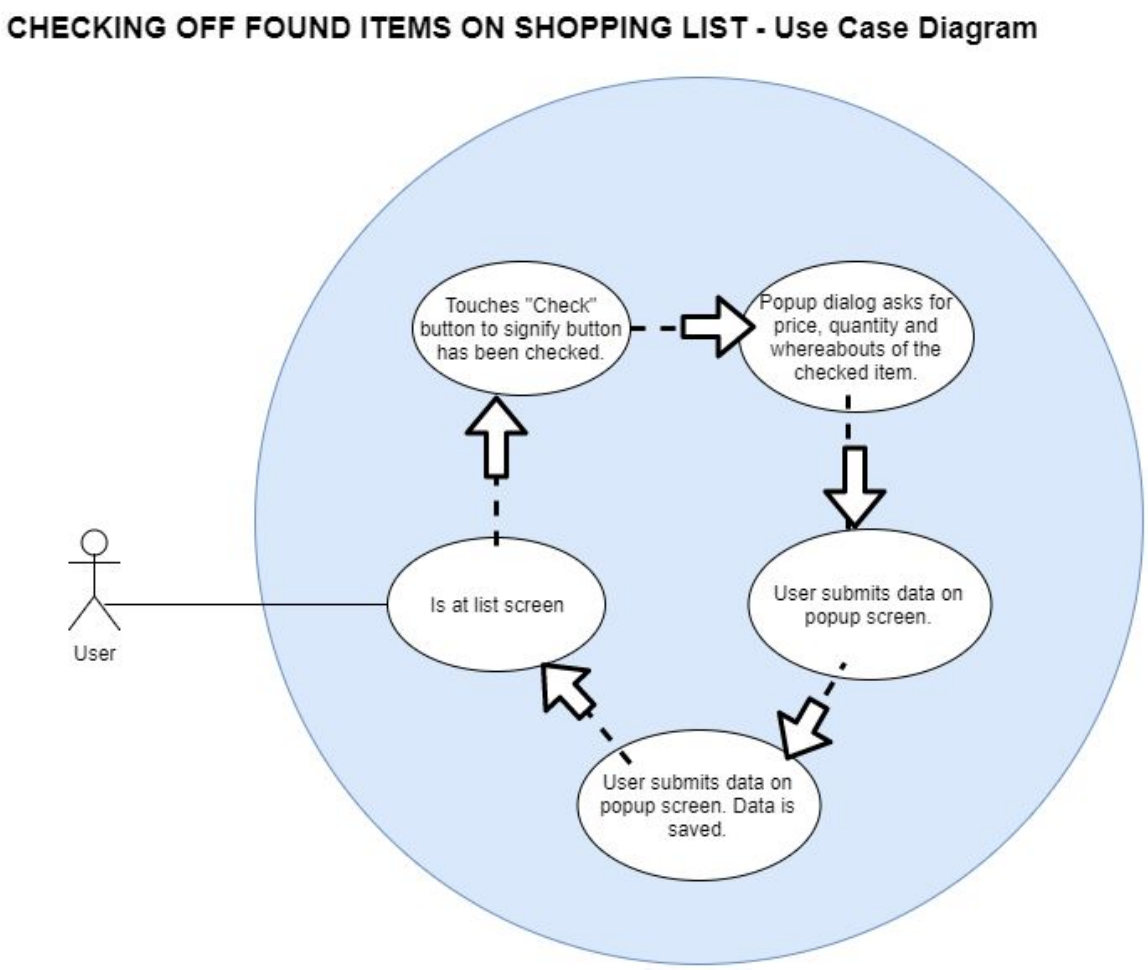
Use Cases:

Considering the previous user requirements at this point it is important to highlight different use cases that users of the system will encounter before we get into the functional description of our system. The selected use cases are the most significant ones considering user interaction one being entering in items before going to the store and then while at the store checking off found items.

ENTERING ITEMS INTO SHOPPING LIST - Use Case Diagram



At the most basic function this is what the proposed application should do for the user. It should allow them to enter in as many items as they would like into their shopping list. This interaction happens while the user is making their shopping list.



This use case is when the user is at the store and will enter in more data about the item when they check it off their shopping list. This shows the circular effect that allows the user to easily provide substantial data about products and their location in very few steps.

Considering the use cases it is important to now describe how we are going to provide these use cases and all of the work going on behind the scene to make this all possible.

Functional Requirements:
The following functional requirements are attached to a specific User Level Requirement where “UR_” corresponds to the appropriately labeled User Level Requirement as defined previously. Using “FRQ_” as a label for the following functional requirements we will define as such. These requirements describe how the system will accomplish certain user requirements as stated above.

UR1-FRQ1: Must be able to enter product data into the application.

- Must be able to add, delete and edit the following information about products:
    - Price of the product, geolocation information such as store address, store name and number of items bought. These data inputs will be collect from easy to use text input fields, a map to which the user can point out a location to, auto collection of geolocation data from the user's phone upon approval of access, checkboxes and submit buttons to store and process the data.
- User must be able to add, delete and edit the location of a store and the name of that store in order to contribute to identifying stores.
    - This will be accomplished by allowing the user to interact with text field inputs and buttons to submit data to the system.

UR1-FRQ2: This system will provide easy to use check boxes so that the Users must be able to check and uncheck items to perform action on.
The checking and unchecking should be easy as a press of a button on the screen with touch. This will allow the user to edit, delete or signify they have picked up the items in their list.

UR2-FRQ1: This system will provide that grocery list items will not store information about whom gave information on them in database.
By not attaching metadata to items in the crowdsourced database even on a data breach information about what the contributors have bought will be given.

UR2-FRQ2: The system will provide a way to store data locally if no internet connection is available by the means of Firebase and the asynchronous API used to synchronize data between the server and all devices using the service.
If there isn't any internet connectivity on the phone then the application must recognize that and try to sync data at the next available time.

UR2-FRQ3: Data transfer will adhere to Firebase/React Native communication API.
By following the Firebase and React Native tutorials on data transfer and adhering to all notices and warnings it will be sure that the app is communicating in an encrypted and safe manner for all types of data to be passed.

UR2-FRQ4: Price suggestion calculations will be done on the users device.
Because the app doesn't communicate data about who gave data on which products or stores all the information on that should only be stored on the user's phone. So that

when the user adds a new item and it does have a lower price and they are within proximity they do get a notification but that by doing this security is thus preserved.

UR2-FRQ5: Geolocation data from the user will not be attached in any way to any specific user instead will be stored as part of items themselves.
This requirement makes it so that we are not holding geolocation information about the user but rather about products within stores and the stores themselves. Geolocation data will be gathered in the following ways:

- Gathered when taking items off the grocery list.
- Gathered when near a common store and to check if the user is near a known store.

UR3-FRQ1: The system will notify users that our app would like to use notifications. Mention where this setting can be found if they deny to allow us access. Do not automatically assume they are ok with this because it could lead to frustrated users. Application should function otherwise independent of this feature even though this feature would depend on the other parts of the application.

UR3-FRQ2: Popup a notification on the users device when a low price appears at a store near them.
When a user has given us permission we can suggest to the user a lower price we found at a store near them. There should only be 5 notifications at most per day.

UR3-FRQ3: Proximity calculation should be done on the user's phone and in the background.
Because data about store locations will be transferred but not user locations the calculations to determine nearby stores should be done on the user's phone in the background so it doesn't interfere with the operation of the device.

UR4-FRQ1: Setup a database in the Firebase platform to save data about products and stores.
This is done so that it can be cloud sourced and can be scaled if necessary. Per our client's request of attaining a crowdsourced grocery list application this is the easiest and safest route.

UR4-FRQ2: Use Firebase API for React-Native in order to establish database connection.
By using this we can ensure reliable transmission of data for future versions of React-Native with Firebase support.

UR4-FRQ3: Firebase API must allow users to login and maintain connection to the Firebase database for every user copy of the application by utilizing the API effectively. This requirement is so that every user will be able to contribute to the crowdsourced database and make sure that connection will be valid.

UR4-FRQ4: Universal database access must be available for all copies of software. This means that all copies of our application must use the save universal database access account to add to the database. This account must still work and be valid even if a credential change is necessary in the future without any update to the application after publishing.

Performance(non-functional) Requirements:
These performance requirements are attached to a specific User level Requirement where “UR_” corresponds to the appropriately label User Level Requirement. Using “PRQ_” as a label for the following performance requirements we will define as such.

UR1-PRQ1: Application must be free of latency.
The user should be able to use the application and not experience significant latency when navigating between screens and entering in information. This is gauged by using less than 10% of the test devices CPU at any given time within our application.

UR1-PRQ2: Transitions must be smooth.
The user should feel like they are naturally turning pages in a book and not roughly navigating to another page. It should be pleasing to the eye to move to another screen to enter in information or do an action.

UR2-PRQ1: Must not consume battery life significantly.
The users must be able to have the application running in the background and it not impact their battery life in order to deliver a really great application.

UR2-PRQ2: Data transfer must be in real time.
Data must transfer, if internet is available, immediately for all users so that all users can instantly take advantage of new data. This is to provide the best pricing and location information on all items and stores.

UR2-PRQ3: Application collects data in specified 5 minute intervals.
This requirement is it satisfy the requirement from our client that the application must not significantly impact users daily use of their device.

UR3-PRQ1: Notifications must be relayed to user in real time.
Notifications must be displayed to the user as soon as possible from when they are created to help the user get notified about pricing near them. This could save the user time and money.

UR3-PRQ2: Notifications must not use significant battery, CPU or memory usage.
This requirement is to again make sure that the application is operating in good performance to not impact the user's device. This application with all of these performance requirements should be able to be ran on most devices regardless of device specs.

UR4-PRQ1: Network access for database updating must be limited to conserve battery.
Avoiding activating the network card on the device is a must to conserve battery so we will sync data at a specific rate that was specified in the functional requirements to provide low battery usage.

UR4-PRQ2: Cloud storage side should be responsive to respect real time requests.
The responsiveness of Firebase should be enough to respond to real time requests in that if a user is within a range of a store and has sufficient internet connection on their device that they get notified in an appropriate amount of time so that they could take advantage of our notification and price updating features.

Ultimately, these requirements could be summed up as an application that is easy to use such that it feels natural and intuitive; specifically, it must make it easier for the user to use than a traditional shopping list and competing mobile applications.

Environmental Requirements:

1. Application must work on mobile devices of varying sizes.
    a. Mobile devices are of varying sizes so in order to reach the maximum population with this system the system must work with different sized screens this will be done by developing with variable screen size in mind via responsive layouts.
2. It is unknown if the system users will have the use of two hands or not in the store or entering items into a list elsewhere.
    a. This is why the application is as easy to use as possible to reduce the possible frustration if both hands are not available. This will be done by suggesting the store they are at if the system knows the location they are at.

3. Application must work under the basis of touch input.
    a. Since all mobile devices rely on touch to select items on the screen and enter data. This will be accomplished by following guidelines within React-Native to allow for all elements defined to be touch compatible.

Now that it is clear what the system being developed will do, how it will do it, and what performance you can expect from the system let's talk about some risks with this proposed solution in mind and how these risks are being mitigated to provide you the best solution possible.

Potential Risks:

Some of technical risks are as follows:

- Depreciation of software due to new versions of the software being released. This can break the current codebase we have where this new version of software A don't communicate anymore with software B, until an update for software B also comes out. This can cause issues when updating software, but the current app should remain unchanged.
- Building a database that can hold all of the crowdsourced information. If the app becomes popular the team will have to consider a premium plan with firebase because the free version could potentially be overloaded by the information our app will be gathering from users.
- Being able to properly store and compare geolocation data for the appropriate stores, such as if user A entered (-100 81 100) as the location of a store then user B is relatively close to that location the program needs to recognize that, along with dealing with the fact of two stores being potentially really close to each other.

Some of the user risks are as follows:

- Users not wanting to switch to our app because they are to comfortable with their current app choice.
- Integrity of data entered by users. We will have to either use a trust based system or some reporting type system that tells if a given user is entering valid data because inaccurate data will drive users away from using the app.

Project Plan

The major milestones for project success include: 1. Create our Firebase Database, 2. Create our list UI prototype with React Native, 3. Integrate our geolocation API that will record GPS coordinates and store it in our firebase database, 4. Create our price comparison interface prototype, 5. Create an interface to for product location entry, 6. Implement pathfinding algorithm and visualization on store layout. Notice that

the time spent building the now defunct Realm.io prototype is incorporated into the schedule as well. Additionally, a buffer of approximately two weeks is included in the timeline to accommodate any setbacks or unexpected complications.

Figure 2 visually organizes the project timeline.

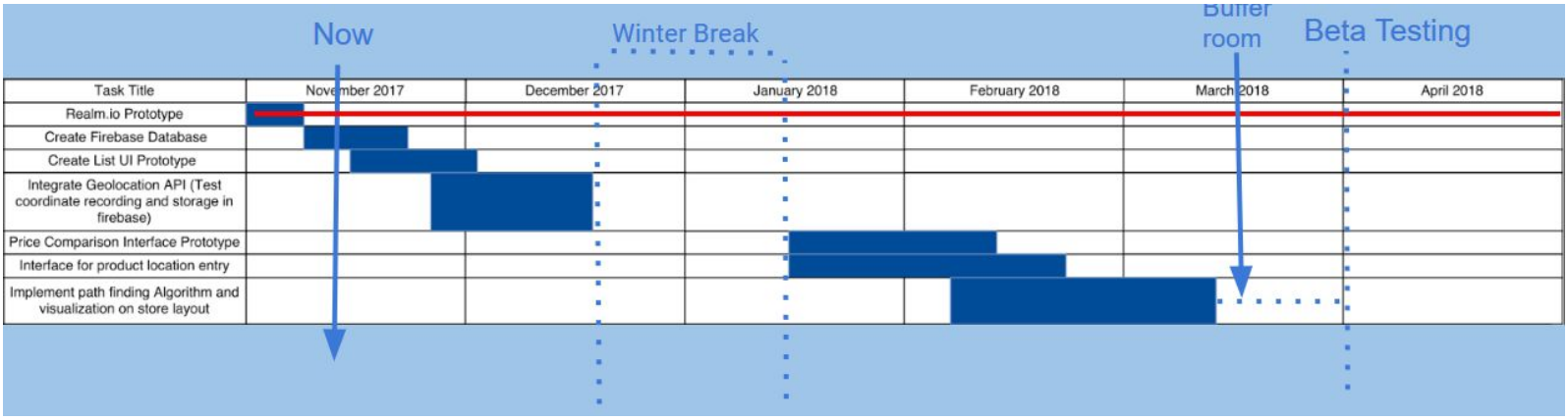


Figure 2.

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. Gnosis Solutions plans to solve this problem with a mobile application that makes use of crowdsourcing that feeds a shared database that is functional with and without an internet connection. This application will also include a streamlined and intuitive user interface that will propel it above competing software on the market. The requirements in this document outline goals for this project, as well as where obstacles may occur in achieving them. The lack of change in the project design since its inception is indicative of the accuracy with which Gnosis Solutions has pursued the best tools and technologies suited to solve the given problem. Imminent prototypes will signal the shift from idea to reality for the next age of shopping list applications.