Team Name : Team Alpha

Software Project Name : Grocer

Team Members :

Project Manager: Jon Stephens

Quality Assurance: Cameron DeVaul

Technical Lead: Travis Tibbetts

Final Software Document: Requirements, Design, Implementation and Test Plan

Date: 12/06/2018

Version: 2.0

Grocer is a grocery list created with the intention of empowering the user to make smart purchases based on their diet restrictions.

Contents

[1. Introduction 1](#_Toc410643945)

[1.1. Project Objective 1](#_Toc410643946)

[1.2. Procject ScoPe 1](#_Toc410643947)

[1.3. Success Criteria 1](#_Toc410643948)

[1.4. Collaboration with Stakeholders 1](#_Toc410643949)

[2. Project Plan 2](#_Toc410643950)

[2.1. Work Breakdown Structure (WBS) 2](#_Toc410643951)

[2.2. Project Resources 2](#_Toc410643952)

[2.3. Responsibility Matrix 2](#_Toc410643953)

[2.4. Gantt Chart 2](#_Toc410643954)

[2.5. Pert Chart 2](#_Toc410643955)

[2.6. Cost Estimation 2](#_Toc410643956)

[2.6.1. Function Point Estimation 2](#_Toc410643957)

[2.6.2. Lines of Code Estimation 2](#_Toc410643958)

[2.6.3. Cost Estimates 2](#_Toc410643959)

[2.7. Risk Plan 2](#_Toc410643960)

[2.8. Project Monitoring and Control Mechanisms 2](#_Toc410643961)

[3. Requirements/Analysis Models 3](#_Toc410643962)

[3.1. Major Software Functions 3](#_Toc410643963)

[3.2. Use Case Diagrams 3](#_Toc410643964)

[3.3. Use Case Descriptions (Fully Dressed) 3](#_Toc410643965)

[3.4. Activity Diagrams 3](#_Toc410643966)

[3.5. Sequence Diagrams 3](#_Toc410643967)

[3.6. Requirements Class Models 3](#_Toc410643968)

[3.7. Prototype Description 3](#_Toc410643969)

[3.8. Data Dictionary 3](#_Toc410643970)

[3.9. Limitations and Constraints 3](#_Toc410643971)

[3.10. Non-functional Requirements 3](#_Toc410643972)

[4. Problems Encountered 4](#_Toc410643973)

[5. Bibliography 5](#_Toc410643974)

# Introduction

## **Project Objective**

The Objective of our application, Grocer, is to make life easier for users by informing them of what exactly they’re eating and whether or not they should be eating it. Grocer does this by informing the user whether or not a product is safe to consume or not, based on allergies, or diets, that the user specifies. Grocer is intended for all audiences that would like to adhere to a specific diet, or have a tool that allows them to easily see content that may concern them within their food.

## **Procject** **ScoPe**

Naturally, Grocer has the ability to become an all-encompassing grocery store application. In the future, Grocer may evolve into this type of application, but for now Team Grocer’s primary objective is to create a prototype, with a limited database, that helps the user abide by a selection of 12 diets. These diets include, but are not limited to, most allergies, veganism, vegetarianism, etc.

## **Success Criteria**

Project Grocer’s current objective will be a success if the following criteria are met; a running application that allows the user to search through a database of products is created. If said application is able to enforce various different diets, including multiple diets, at the same time. Lastly, if this application is able to read barcodes and bring up the correct corresponding item information.

## **Collaboration with Stakeholders**

Collab with Magdalene Louise Bamber

* Magdalene is on board with Grocer, but has a few concerns and things she would like added to the application. She would like for the application to also have a recipe portion. She believes that adding a set of recipes that pertain to each diet/allergy will help those affected by these restrictions shop at the grocery store and give them another reason to continue to use the app. Her main concern is whether or not the database will be populated sufficiently.

Collab with Richard Jacob Wnuk

* Richard likes the application and has no concerns as far as the functionality of the application. His “suggestion” was that the application should suggest recipes based on the cart. If you have Onions, pickles, and rice, then the application should suggest a recipe using some or all of those ingredients.

## **Similar Products**

1. OurGroceries
   1. OurGroceries is the closest application to what Grocer is intended to be. The simplicity of the application is what makes it so attractive and easy to use. Although it holds no information about the items being purchased, the list functionality of OurGroceries is everything you would need from a grocery list application.

* Pros
  + Holds Recipes
  + Remembers Items
  + List Sharing
  + Easy Use
* Cons
  + Bland UI
  + Nothing more than a smart list

1. MyFitnessPal
   1. The best all-around app. MyFitnessPal has all the functionality needed to lose weight effectively and track your micro and macro nutrients. With the extensive database including grocery items and fast food items, there isn’t a better app for tracking your nutrients. The only downside to MyFitnessPal is you have to log everything you eat in order to benefit from it. For those who are committed to losing weight or eating healthier this is no obstacle, however for the casual user this becomes more work than desired.

* Pros
  + Calorie Tracking
  + Expansive Food Database
  + Barcode Reader
  + Remembers Items
  + Food / Activity Suggestions
  + Progress Tracker
  + Social Media Interaction
* Cons
  + High upkeep

1. AnyList
   1. AnyList is a list application that offers higher functionality than most grocery list application. AnyList’s meal planning and ability to organize grocery lists, and food within those grocery lists, is superb. The only downfall to this application is that all the functionality and customizability that AnyList offers is only available with a premium subscription. Most people won’t pay $1 for a helpful application, let alone AnyList’s $7.99 yearly subscription.

* Pros
  + Recipe Suggestion (Web Browser)
  + Meal Planning (Premium)
  + High Customizability
* Cons
  + Account Creation feels unnecessary for free version
  + Premium subscription
* Things Learned :
  + **MyFitnessPal** - Having a highly capable application is great in theory, but if it becomes over encumbering to get the use out of the app then users will stop using it after a while.
  + **AnyList** -Offering a free bare bones application with minimal functionality, while simultaneously offering a premium version of the application with high capability, customization, and ease of use makes the user feel like they’re being cheated on initial usage of the free application.
  + **OurGroceries**  - Simple applications that are easy to use are the optimal route to go. Although lacking in functions, these apps provide enough use to warrant a download and do what the user was wanting in the first place.

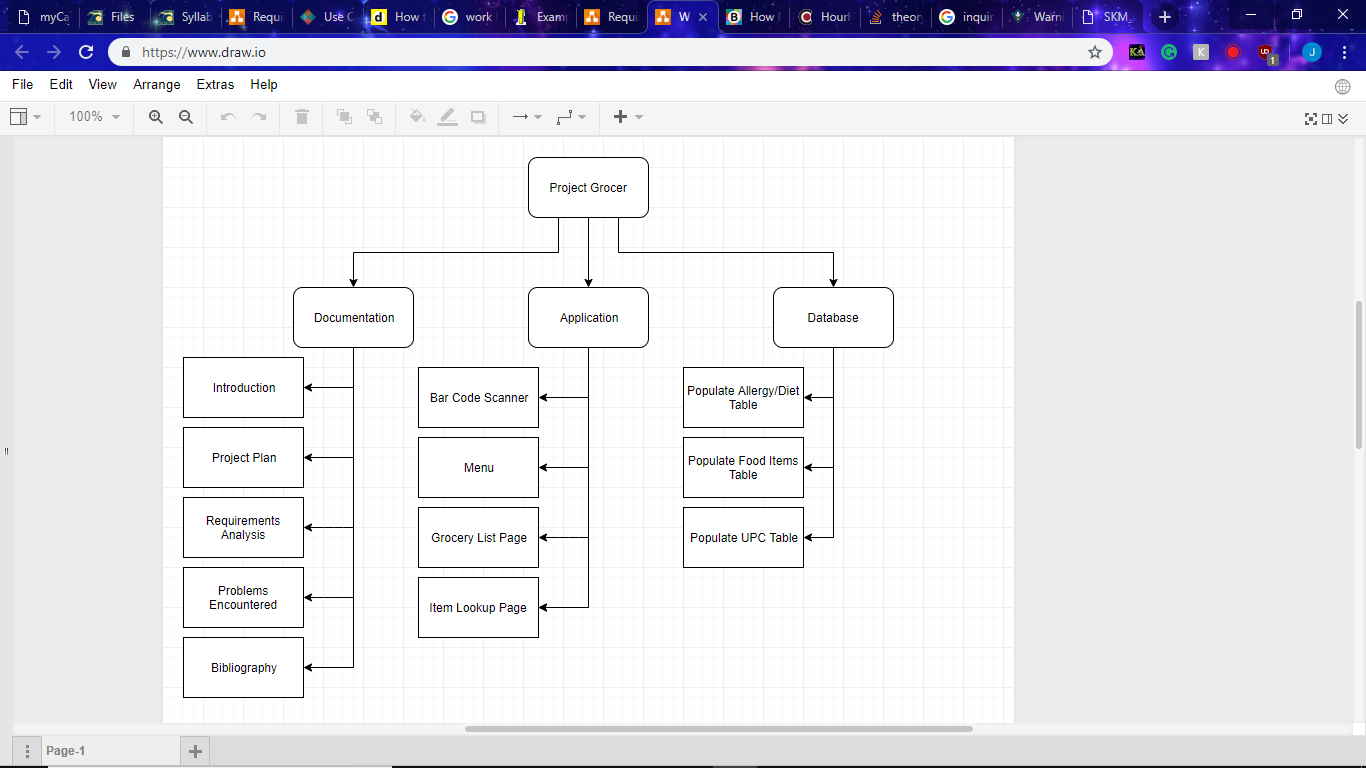
# Project Plan

## **Work Breakdown Structure (WBS)**

Project Grocer

1. Documentation
   1. Introduction
   2. Project Plan
   3. Requirement Analysis
   4. Problems Encountered
   5. Bibliography
2. Application
   1. Bar Code Scanner
   2. Menu
   3. Grocery List Page
   4. Item Lookup Page
3. Database
   1. Populate Allergy/Diet Tables
   2. Populate Food Items Table
   3. Populate UPC Table

## **Work Breakdown Structure (graphic)**



## **Project Resources**

SQLite

Android Studio

Draw.io

Microsoft Word

Microsoft Excel

## **Responsibility Matrix**

## Accuracy Assessment

* More or less we stuck to this responsibility matrix, however this was not our initial plan. Our initial plan was that we would all work on each part together, despite that we ended up all doing one individual all-encompassing task that involved quality assurance, organization and risk management for their own separate part, Jon Stephens with Documentation, Cameron DeVaul with app development, and Travis Tibbetts with database development. Travis Tibbetts did assist with documentation. We all discussed risk management and organization throughout each week and when we met however.

R – Responsible (Does Task)

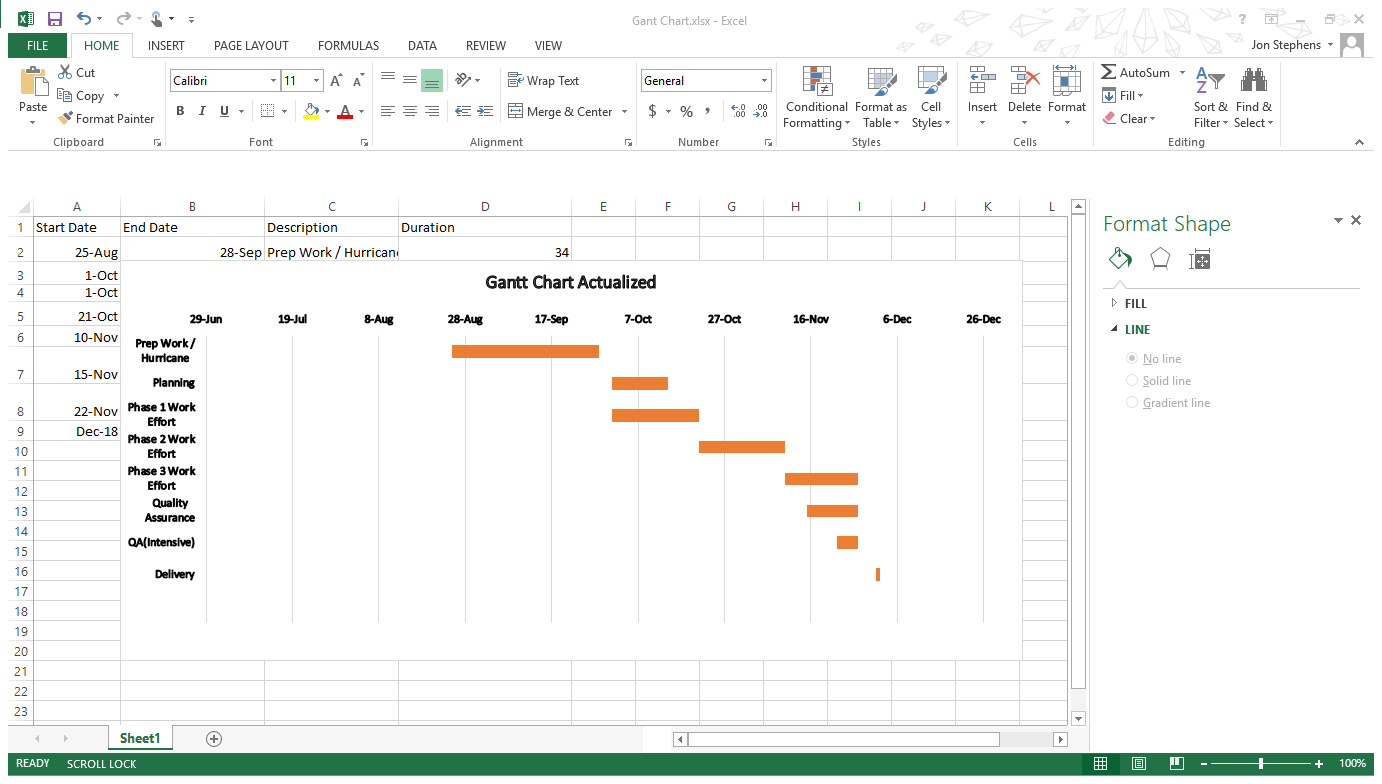
A – Accountable (Ultimately Responsible)

C – Consulted (Provide Feedback)

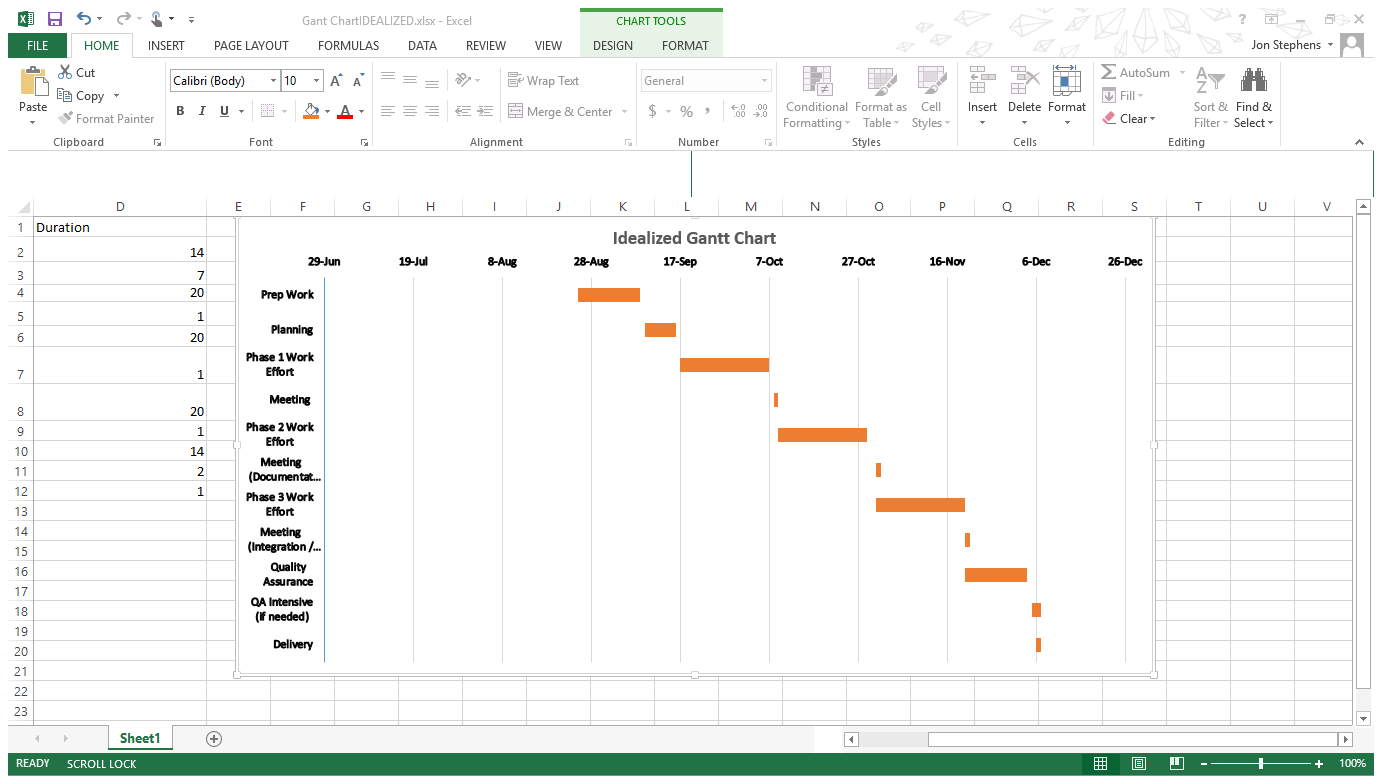
I – Informed (Knows of)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Documentation | App Dev. | DB Dev. | QA | Risk Management | Project Organization |
| Cameron | I/C | R/A | I/C | R/A | R/C | A/R/C |
| Travis | I/C | I/C | R/A | C | R/C | A/R/C |
| Jon | R/A | I/C | I/C | I | A/R | A/R/C |

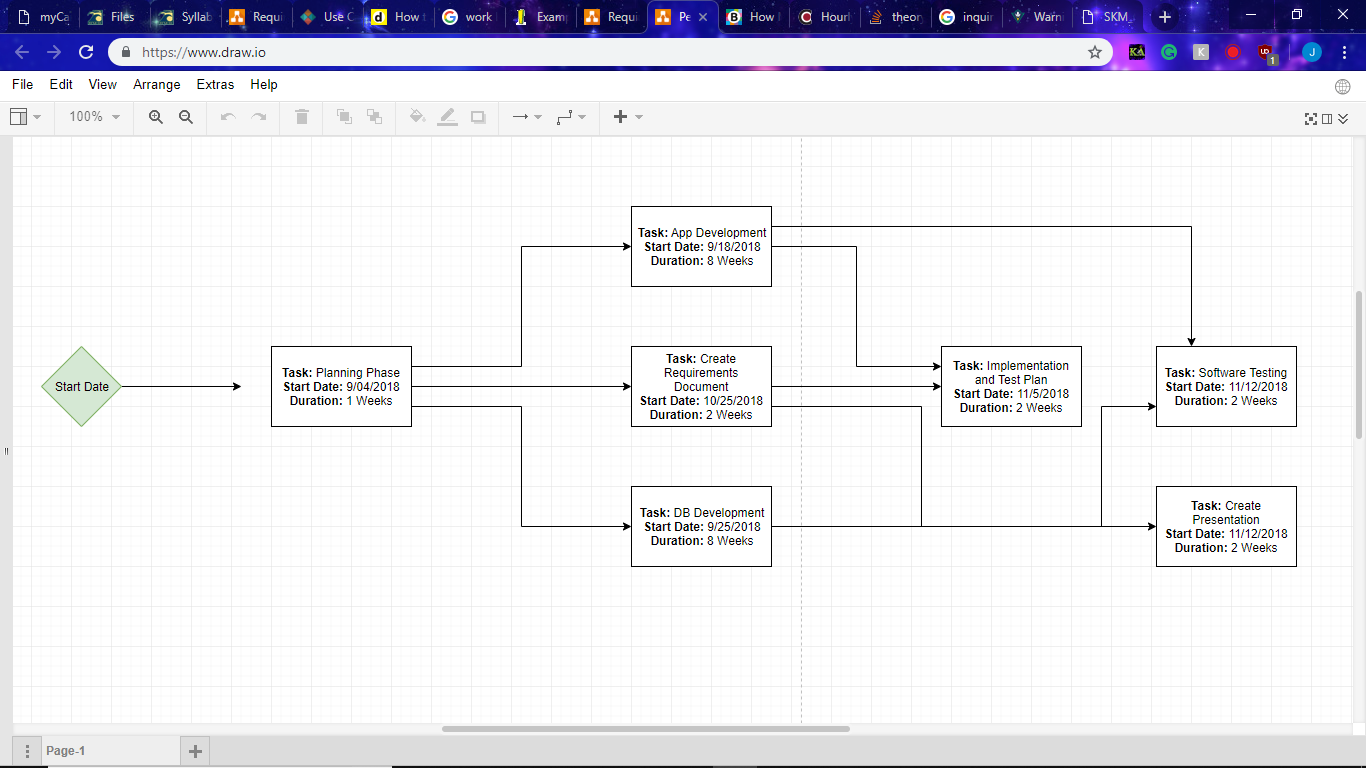
## **Gantt Chart (ACTUALIZEDS)**



## **Gantt Chart (Idealized)**

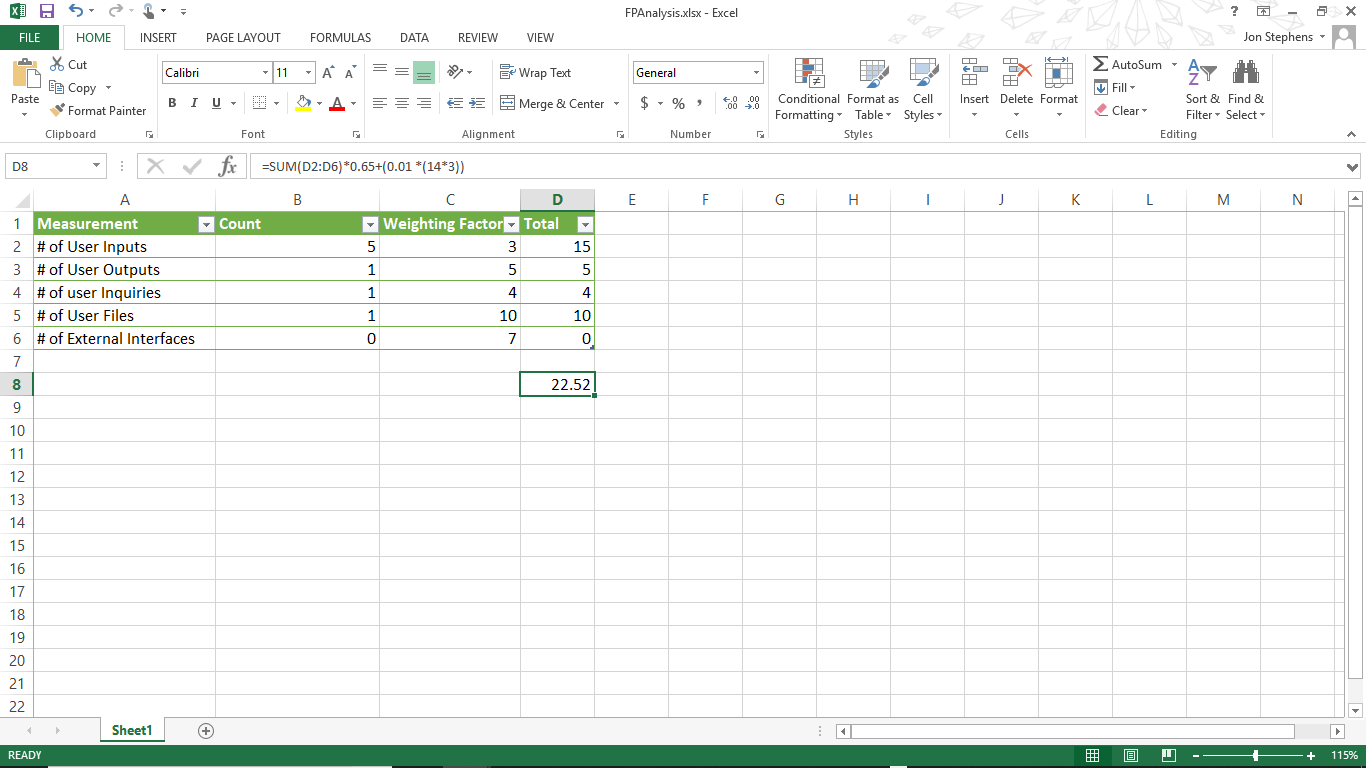


## **Pert Chart**

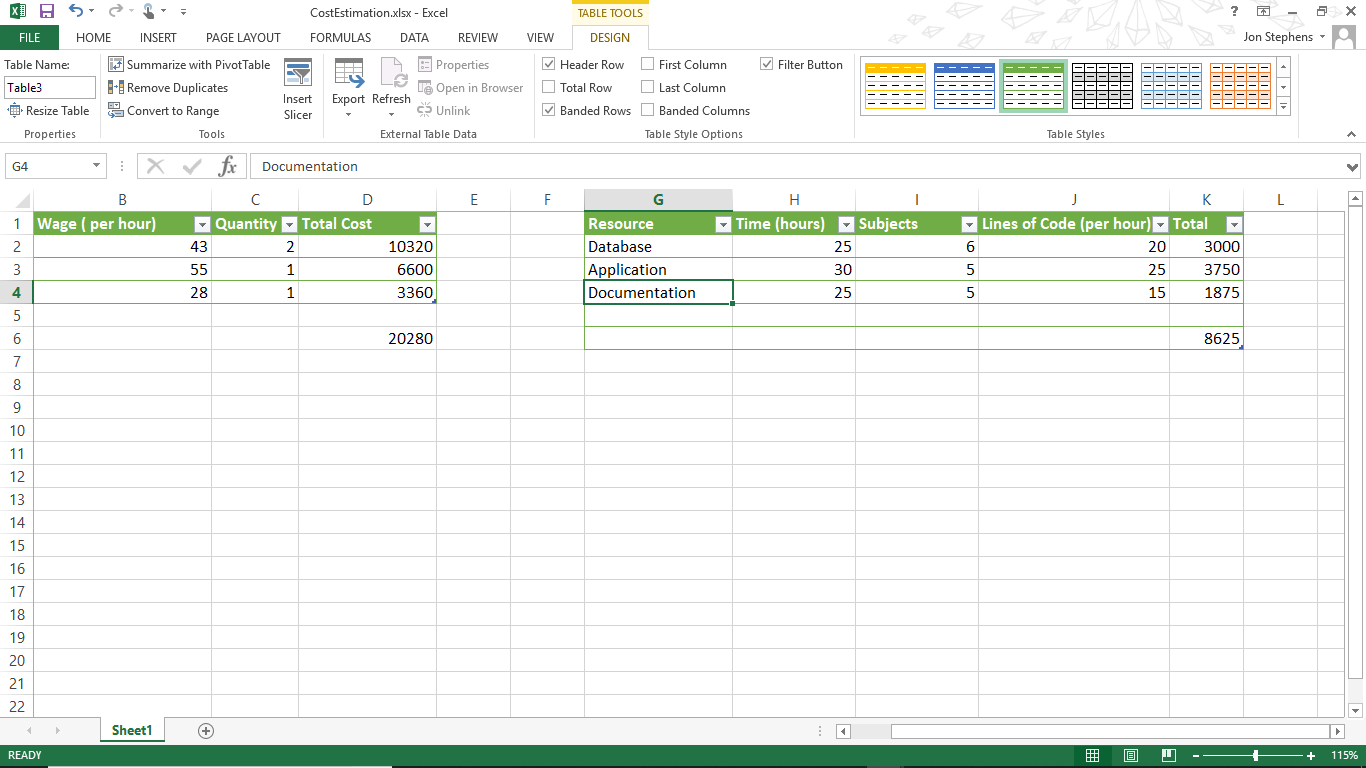


## **Cost Estimation**

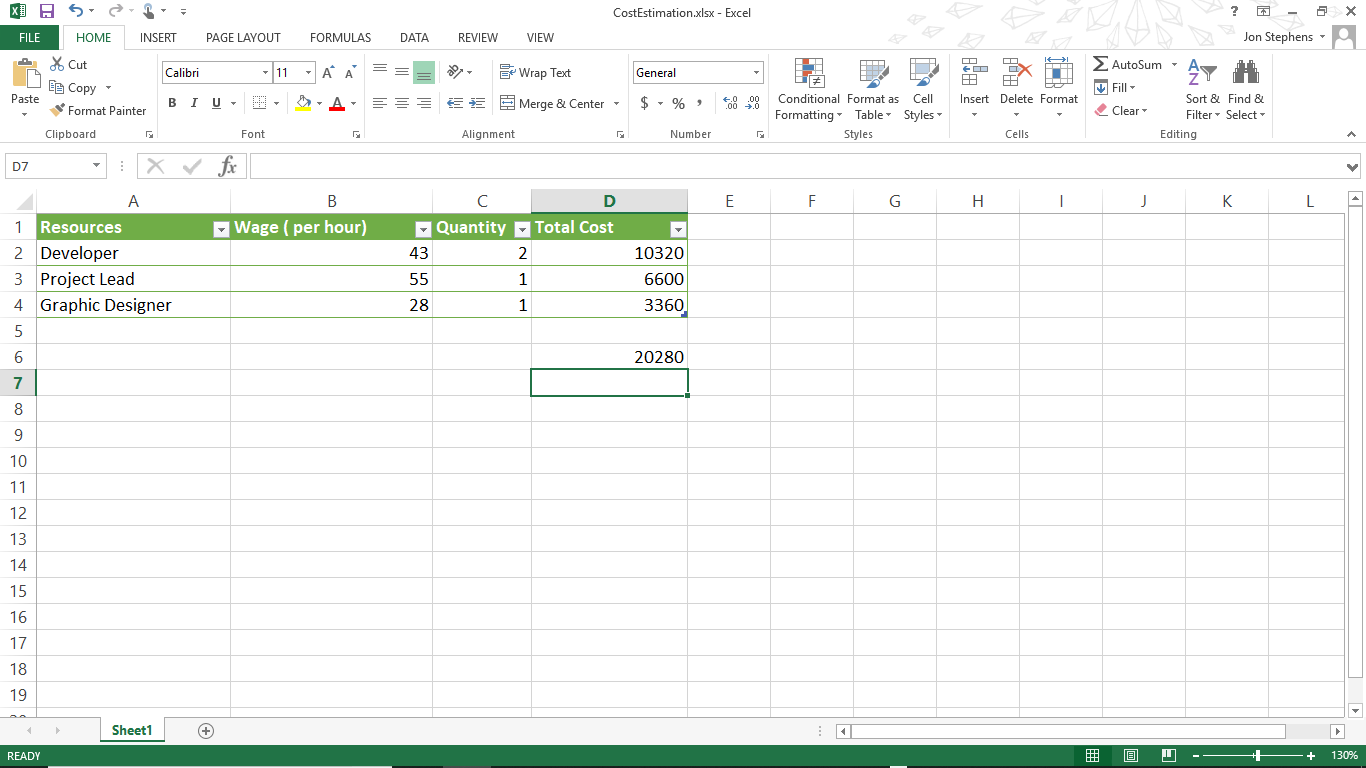
### Function Point Estimation



### Lines of Code Estimation



### Cost Estimates



### Final Evaluation of Cost

As far as the lines of code estimation, these values, depending on how you view them, were mostly underestimated. All three facets, database, application, and documentation, has taken more than the estimated time given, however if you view it as pure time worked on the subject they would be more accurate. Documentation has taken approximately 30 or 40 hours and, excluding the graphs, is at 4500 words within the document, lines of code is not equivalent to words within a document but each graph approximately takes 20 to 30 minutes and with 15 graphs that equates to 105 extra lines of code. On the application side of things, there is 1400 lines of code, 80 to 100 lines per class and 16 classes, and more coming. The database has 175 entries, which is less than initially calculated, however it is more than enough entries for a prototype. It’s difficult to say more effort would be given despite the hurricane, the unfortunate truth of it all is all classes asked for the same work load to be completed with a smaller time span, causing project based classes, which seniors mostly have, to all have a plethora of projects due within small increments of each other, not to mention the stress of working on all of those projects simultaneously and keeping up with extracurricular activities. To say more work would have been done may not be the whole truth, but more relaxed and unrushed work would have been done.

## **Risk Plan**

|  |  |  |  |
| --- | --- | --- | --- |
| Actions | When? | Who? (Main person in Charge) | Comments |
| Documentation | 12/06/2018 | Jon Stephens |  |
| Database (populated) | 11/25/2018 | Travis Tibbetts |  |
| Database (implemented) | 12/1/2018 | Travis / Cameron |  |
| Prototype running | 12/06/2018 | Cameron DeVaul |  |
| Presentation (PowerPoint)? | 12/06/2018 | All Members |  |

## **Project Monitoring and Control Mechanisms**

**Cost Control** – We aren’t paying anyone.

**Scope Control** – Grocer has been dialed down to a functional prototype in effort to make it more manageable. The initial schematics of Grocer involved it having full nutritional facts including calories, sodium, carbohydrates, etc. However, for the Grocer prototype it is functioning primarily as an allergy / diet grocery list control mechanism.

**Schedule Control** – There are multiple moments throughout the semester we have scheduled for meetings to check in on the progress of everyone. Aside from that, we’ve decided that we all can assist with each other’s portion of the project if need be.

**Monitoring** – Similar to the schedule control, we are all meeting regularly to see our progression with our portions.

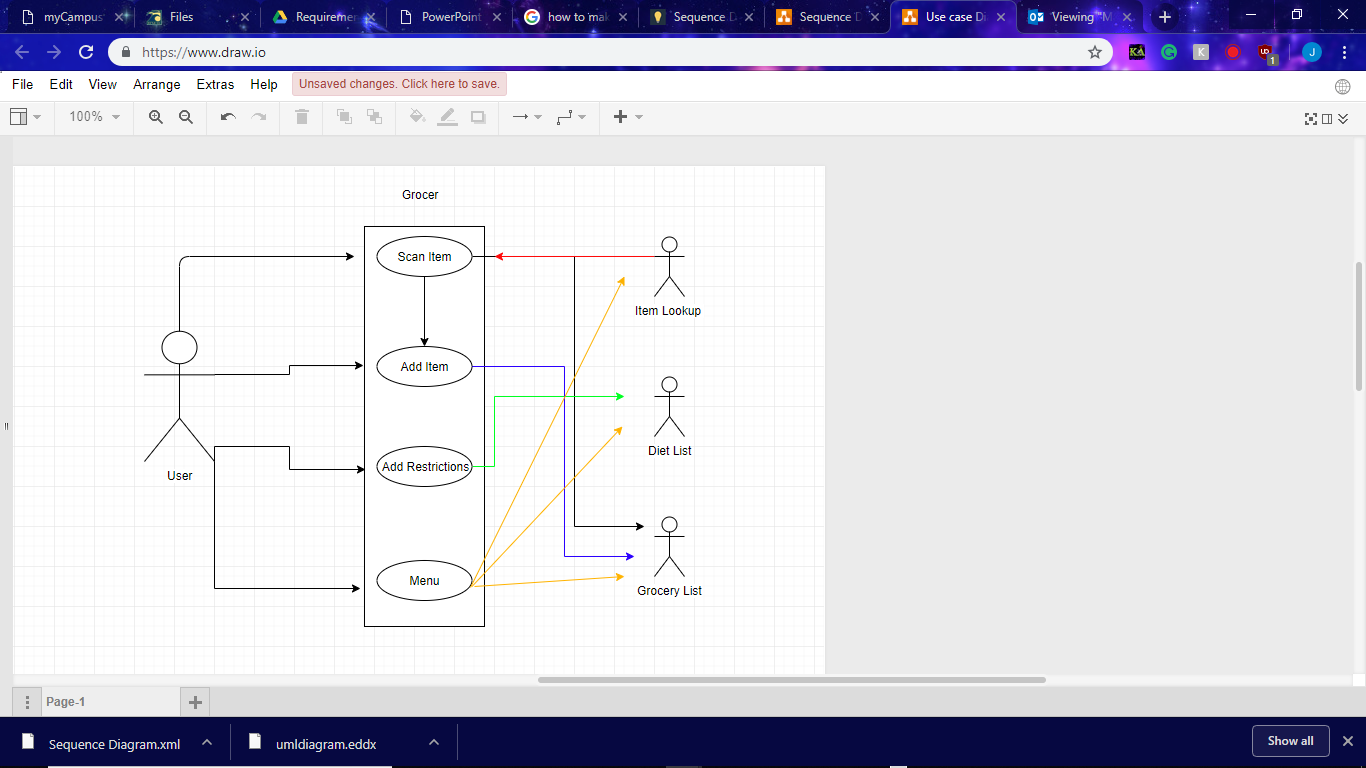
**Quality Control** – By minimizing the scope of the project we are allowing ourselves more time to check the database, documentation, and application for errors. On top of that, due to our regular collaboration we are more easily able to hold each other accountable for creating quality content.

# Requirements/Analysis Models

## **Major Software Functions**

The primary functions of the Grocer app is to allow the user to retrieve health information about the food products they may purchase in the grocery store. The user will be able to choose allergies and diets that they require or want to live by. The user will be able to create a shopping list from a database of food items. The list will hold the food items for the user, and if a food item has a conflict with the user’s specified diet or allergy, the food item in the list will be flagged. The app will also have a barcode function to scan items in the store. If the item is not available in the database, the user will be able to add the item to the database. The user can enter some simple information such as the food item name, and any ingredients that set off a flag. Once the item is in the database, the food item can then be scanned to gather information about it, or even added to the user’s shopping list. These functions are made to make health information about food products easier to understand and more accessible. The app is designed to help individuals with their lifestyle choices.

## **Use Case Diagrams**



## 

## **Use Case Descriptions (Fully Dressed)**

|  |  |
| --- | --- |
| **ID:** | **UC-001** |
| **Title:** | Scan Item |
| **Description:** | This use cases objective is to be able to scan an items barcode and add it to the shopping list. |
| **Primary Actor:** | User |
| **Preconditions:** | App must be on the item lookup page and the scan item button must be pressed. |
| **Post sconditions:** | The barcode will be scanned and a window will pop up that shows the items nutritional facts. |
| **Main  Success Scenario:** | 1. The “Scan Item” button is pressed. 2. The item is found within the database. 3. A page shows up showing the items “flags” if they exist. |
| **Extensions:** | Error Case: If the item is not found within the database  Error Case: If the item doesn’t show the items contents |
| **Frequency of Use:** | Frequency is based on the user. |
| **Status:** | Finished |
| **Priority[1-5]:**  (1 being low priority) | 3 |

|  |  |
| --- | --- |
| **ID:** | **UC-002** |
| **Title:** | Add Item |
| **Description:** | This use cases objective is to be able to add an item to the grocery list. |
| **Primary Actor:** | User |
| **Preconditions:** | App must be on the item lookup page. |
| **Postconditions:** | An item is added to the grocery list. |
| **Main  Success Scenario:** | 1. User either searches for an item or scans barcode. 2. A page shows up showing the items “flags” if they exist. 3. User either adds or doesn’t add the item to the grocery list. |
| **Extensions:** | Error Case: If the item is not found within the database  Error Case: If the item doesn’t show the items contents  Error Case: If the item isn’t added to the grocery list |
| **Frequency of Use:** | This will be the most used function of this application. |
| **Status:** | Finished |
| **Priority[1-5]:**  (1 being low priority) | 5 |

|  |  |
| --- | --- |
| **ID:** | **UC-003** |
| **Title:** | Add Restriction |
| **Description:** | This use cases objective is to be able to add a restriction to the grocery list. |
| **Primary Actor:** | User |
| **Preconditions:** | App must be on the diet page. |
| **Postconditions:** | An item is added to the grocery list. |
| **Main  Success Scenario:** | 1. The user selects a diet or allergy. 2. That selection then raises flags whenever items that pertains to it is added to the grocery list. |
| **Extensions:** | Error Case: if the incorrect restriction is added  Error Case: if the restriction doesn’t raise flags to the items it should  Error Case: if the page doesn’t allow you to select multiple values |
| **Frequency of Use:** | Moderate- High Usage – this is one of the primary functions of the application, however it isn’t needed by all users. |
| **Status:** | Finished |
| **Priority[1-5]:**  (1 being low priority) | 5 |

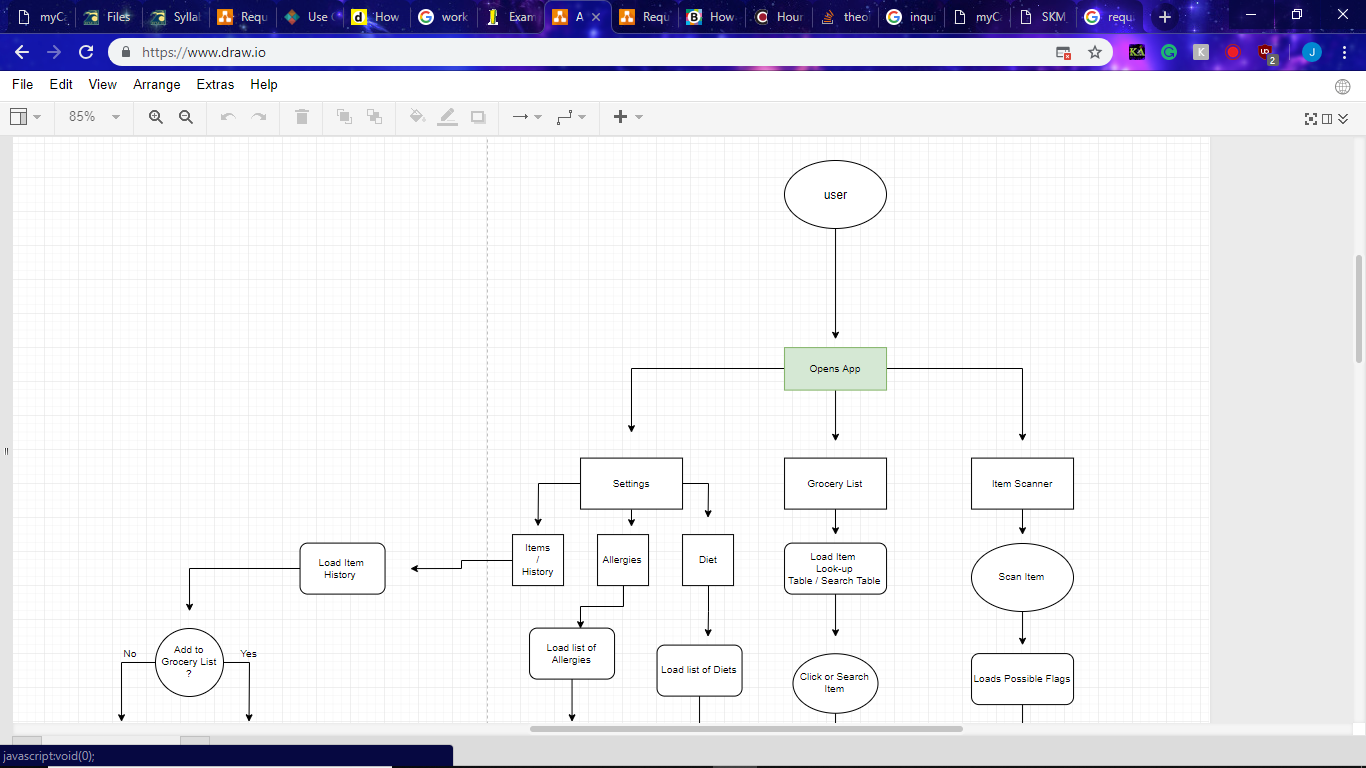
|  |  |
| --- | --- |
| **ID:** | **UC-004** |
| **Title:** | View Grocery List |
| **Description:** | This use cases objective is to be able to add a restriction to the grocery list. |
| **Primary Actor:** | User |
| **Preconditions:** | App must be open and the Grocery list from the menu must be pressed. |
| **Postconditions:** | The grocery list is viewable. |
| **Main  Success Scenario:** | 1. The grocery list button is selected. 2. The grocery list appears. |
| **Extensions:** | Error Case: The grocery list doesn’t appear.  Error Case: The grocery list shows incorrect items. |
| **Frequency of Use:** | Moderate- High Usage – this is one of the primary functions of the application, however it isn’t needed by all users. |
| **Status:** | Finished |
| **Priority[1-5]:**  (1 being low priority) | 5 |

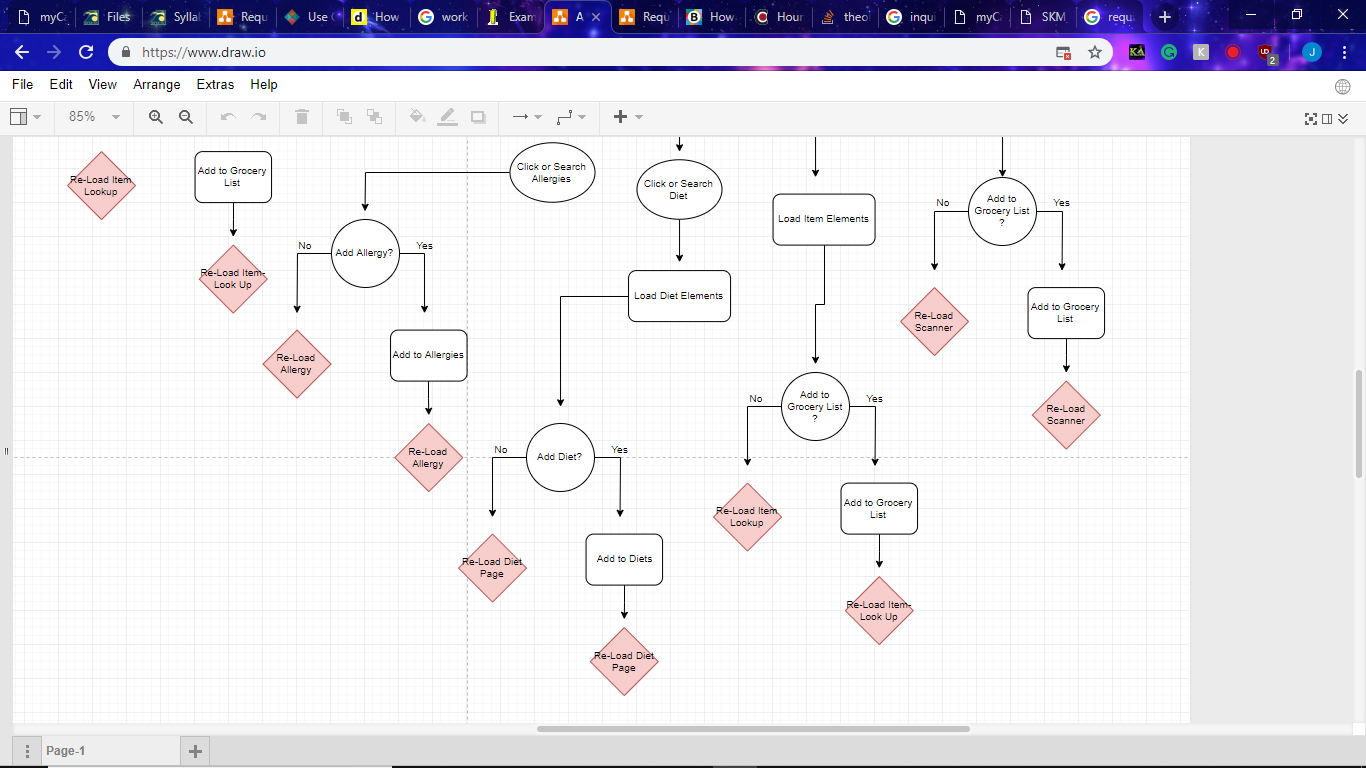
|  |  |
| --- | --- |
| **ID:** | **UC-005** |
| **Title:** | Remove Item |
| **Description:** | This use cases objective is to be able to remove an item from the grocery list. |
| **Primary Actor:** | User |
| **Preconditions:** | There must be an item, or items, in the grocery cart that can be removed. |
| **Postconditions:** | An item is removed from the grocery list. |
| **Main  Success Scenario:** | 1. The item is removed. |
| **Extensions:** | Error Case: The item is not removed. |
| **Frequency of Use:** | Low - High Usage – this is one of the primary functions of a grocery list, however that doesn’t necessarily mean it’ll be highly used. |
| **Status:** | Finished |
| **Priority[1-5]:**  (1 being low priority) | 5 |

|  |  |
| --- | --- |
| **ID:** | **UC-006** |
| **Title:** | Save Grocery List |
| **Description:** | This use cases objective is to be able to save the grocery list used for future usage. |
| **Primary Actor:** | User |
| **Preconditions:** | A grocery list must be created. |
| **Postconditions:** | A grocery list is saved and can be pulled up for later usage. |
| **Main  Success Scenario:** | 1. The grocery list saves. 2. The grocery list is able to be pulled up for future usage. |
| **Extensions:** | Error Case: The grocery list doesn’t appear.  Error Case: The grocery doesn’t save.  Error Case: The grocery save’s incorrect data. |
| **Frequency of Use:** | Moderate- High Usage – this is one of the primary functions of the application, however it isn’t needed by all users. |
| **Status:** | Unfinished |
| **Priority[1-5]:**  (1 being low priority) | 3 |

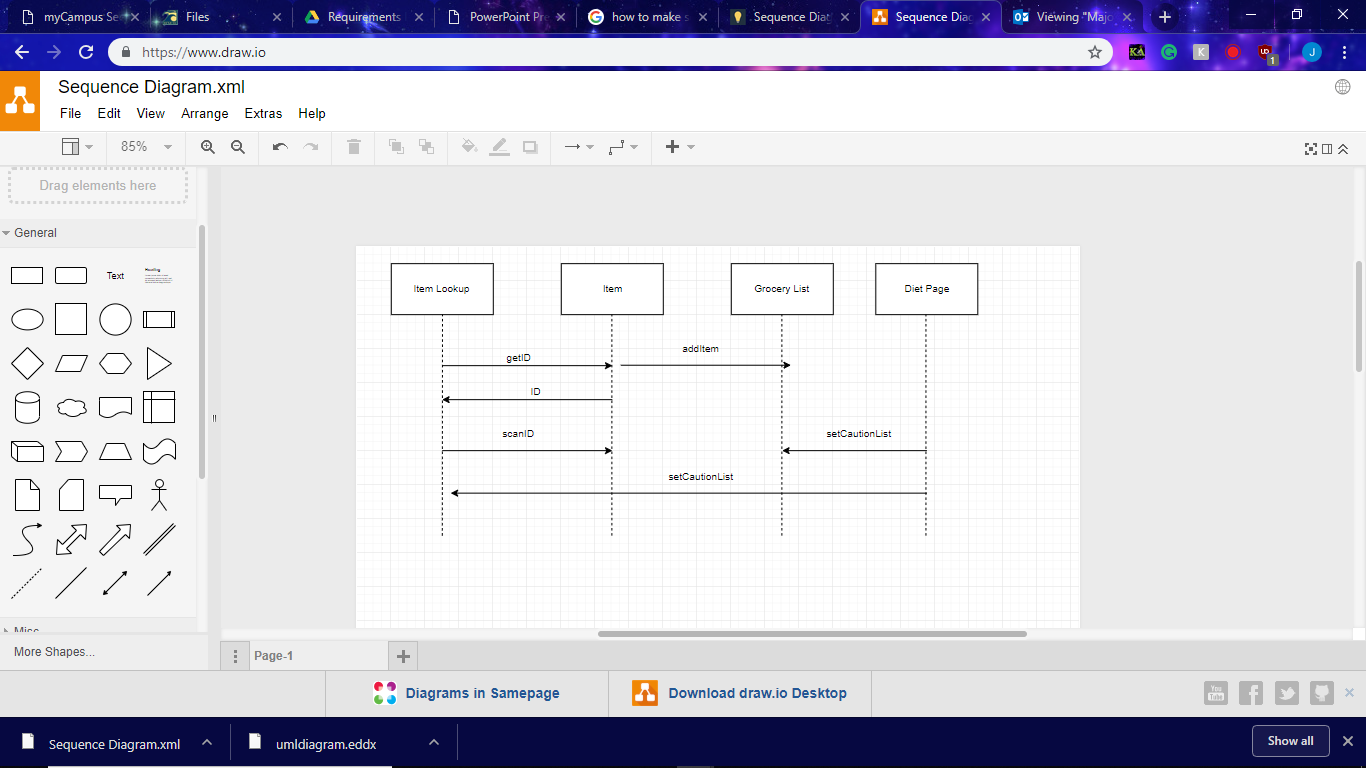
|  |  |
| --- | --- |
| **ID:** | **UC-007** |
| **Title:** | Load Grocery List |
| **Description:** | This use cases objective is to be able to load a grocery list. |
| **Primary Actor:** | User |
| **Preconditions:** | A grocery list must be readily available to load. |
| **Postconditions:** | A grocery list and diet restriction page is loaded into the application |
| **Main  Success Scenario:** | 1. The grocery list loads (and is populated). 2. The Diet page loads (and is populated if needed). 3. The grocery list is editable without affecting the file it was loaded from (unless specifically overwritten). |
| **Extensions:** | Error Case: The grocery list isn’t properly populated.  Error Case: The diet page isn’t properly populated.  Error Case: The grocery / diet page doesn’t load the correct data. |
| **Frequency of Use:** | Moderate- High Usage – this is one of the primary functions of the application, however it isn’t needed by all users. |
| **Status:** | Unfinished |
| **Priority[1-5]:**  (1 being low priority) | 3 |

## **Activity Diagram**





## **Sequence Diagrams**



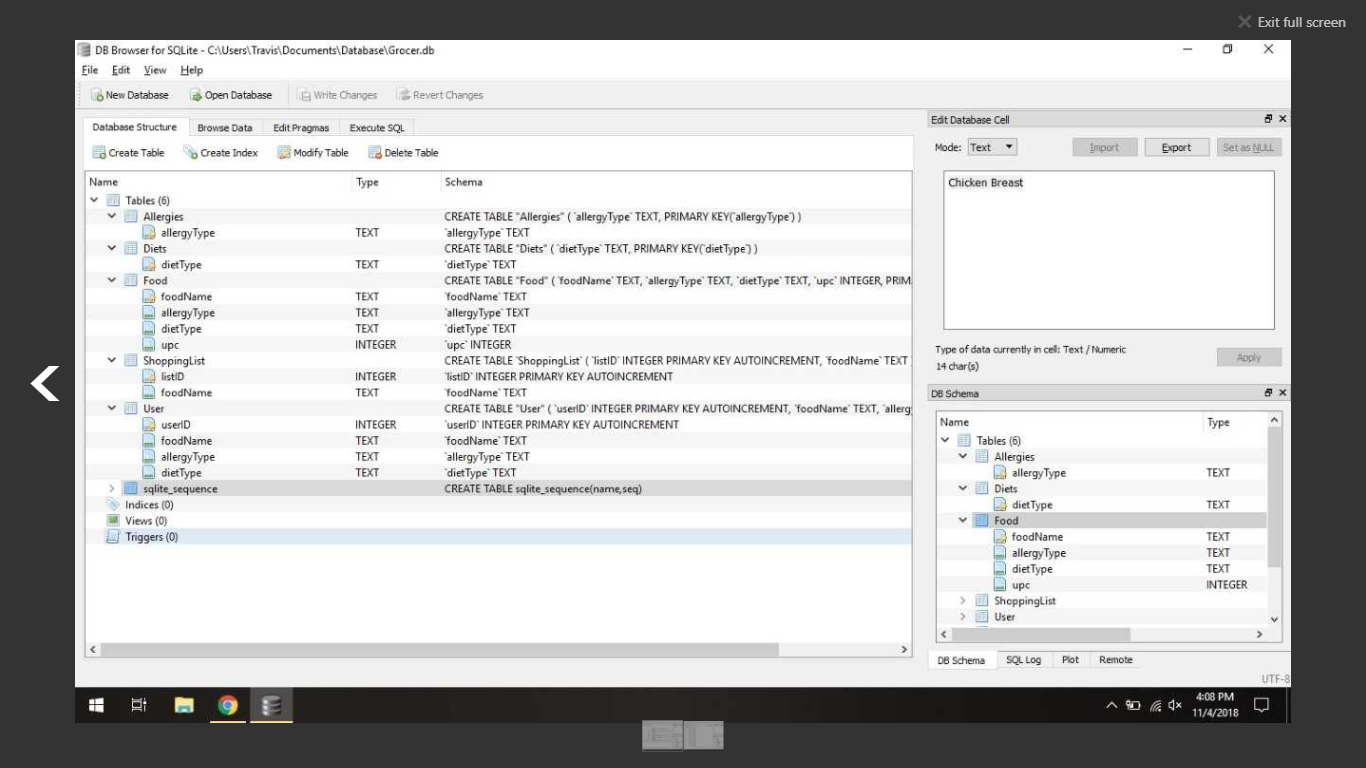
## **Requirements Class Models**

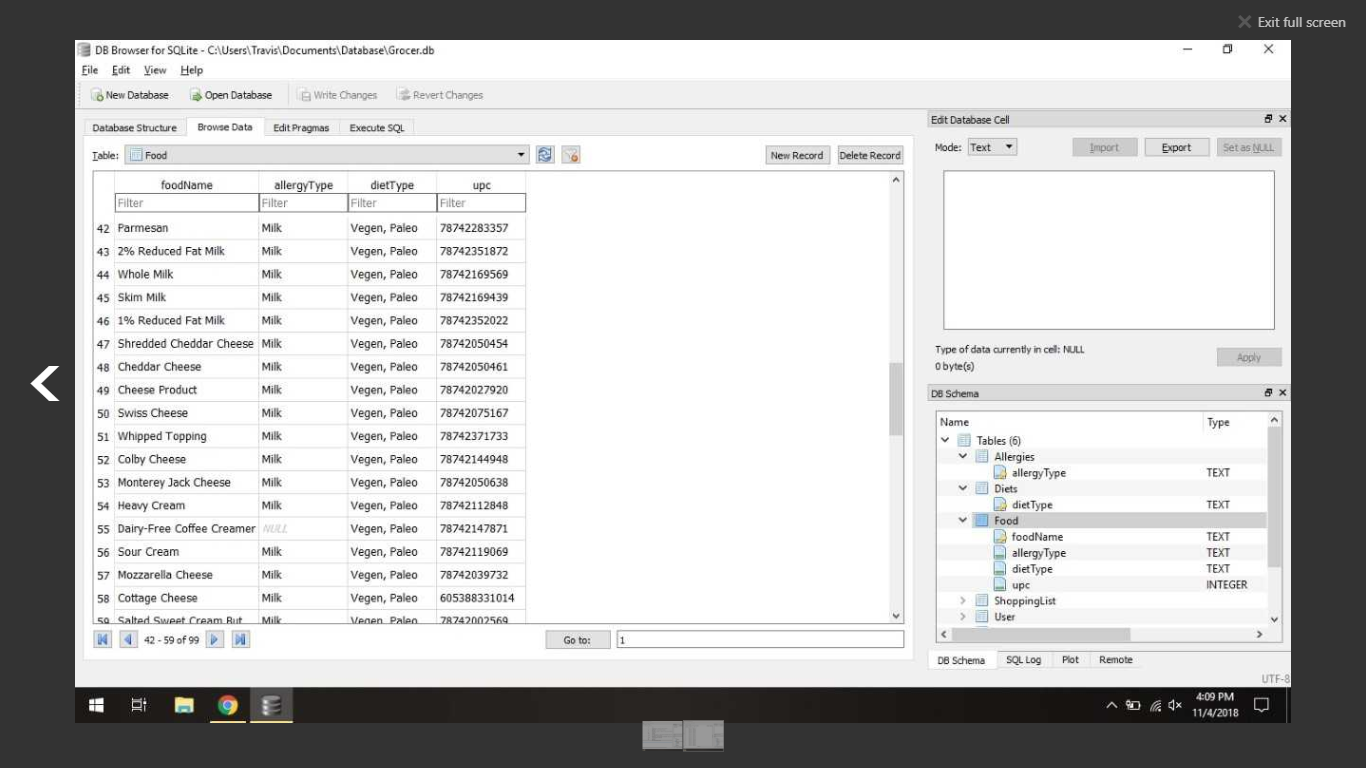


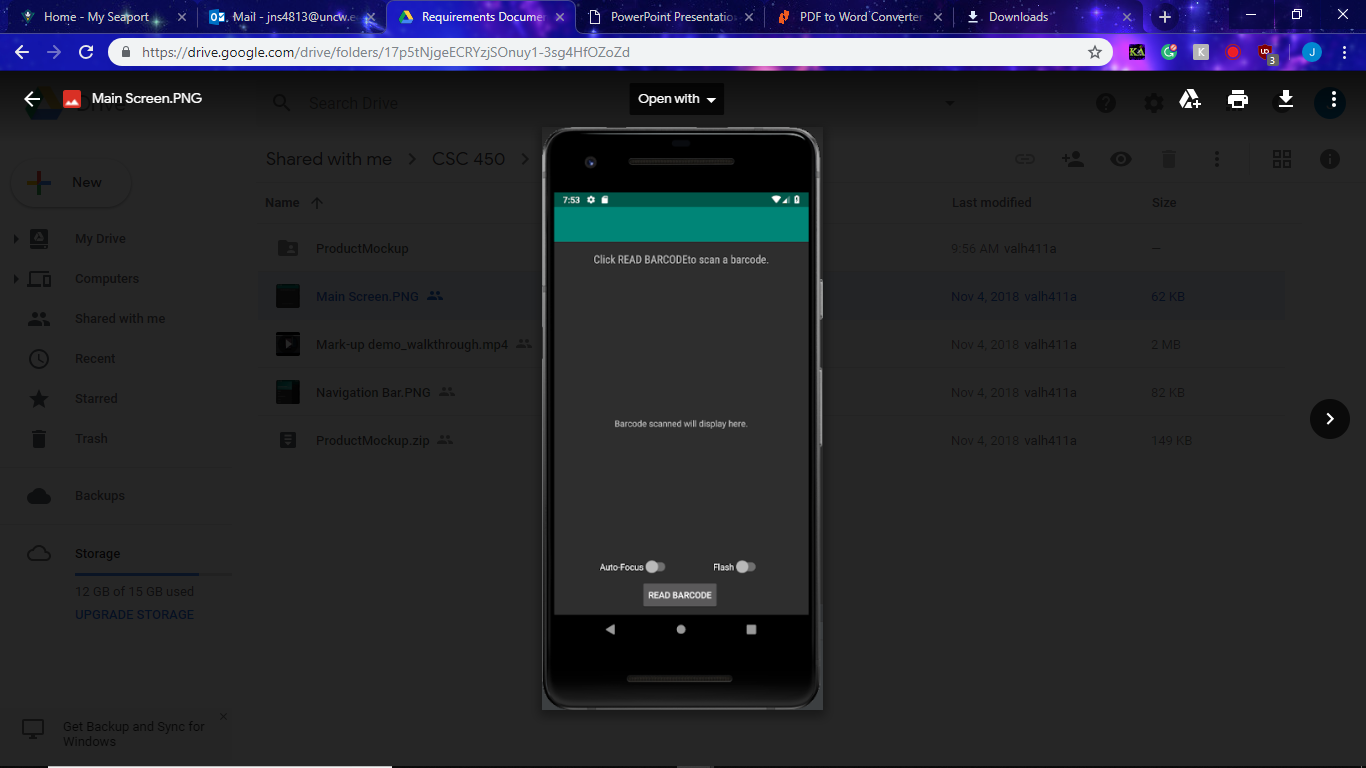
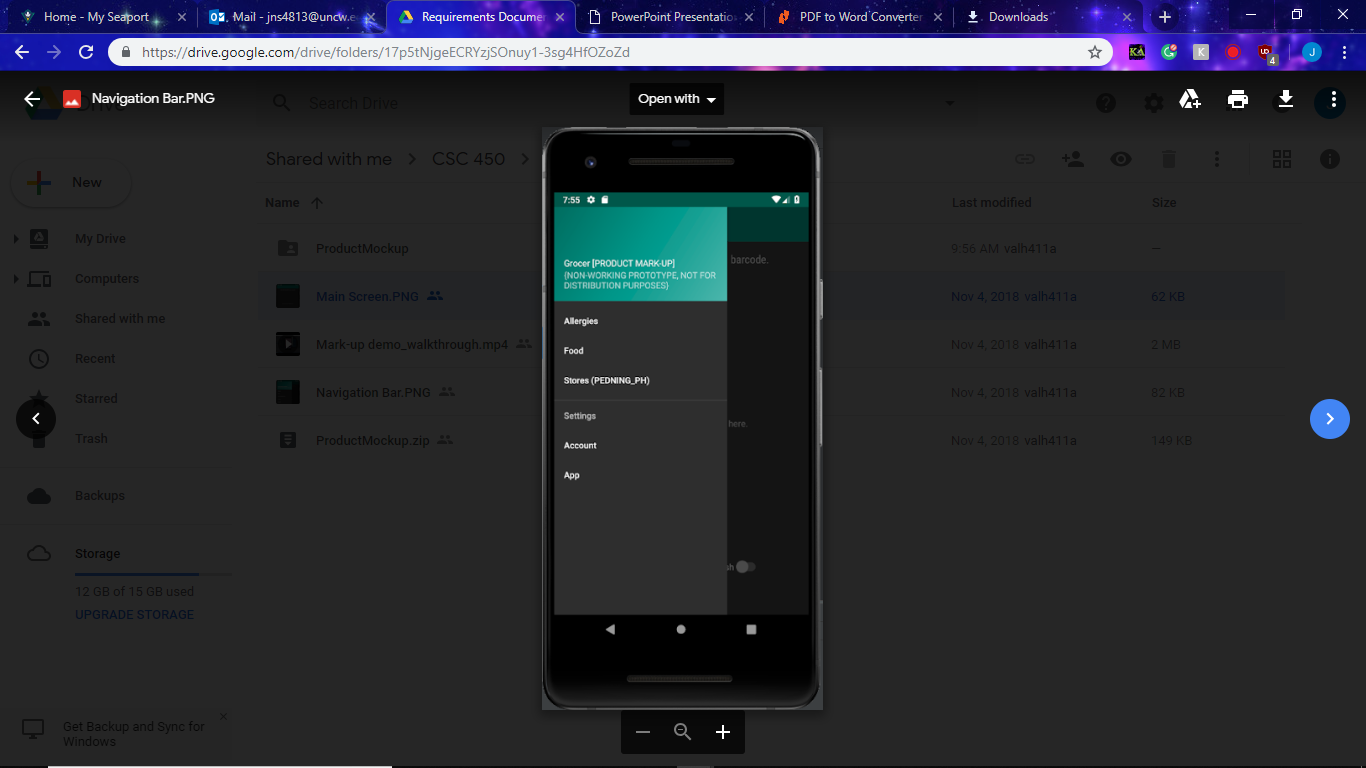
## **Prototype Description**

Currently, the prototype has the basic design of the finished application. It has all the tabs it will need and the necessary functionality to work as a shopping list. The finished version of the prototype will include functionality in all the tabs, having a grocery list page, an item lookup/ scan page and a setting page where you set your diets.

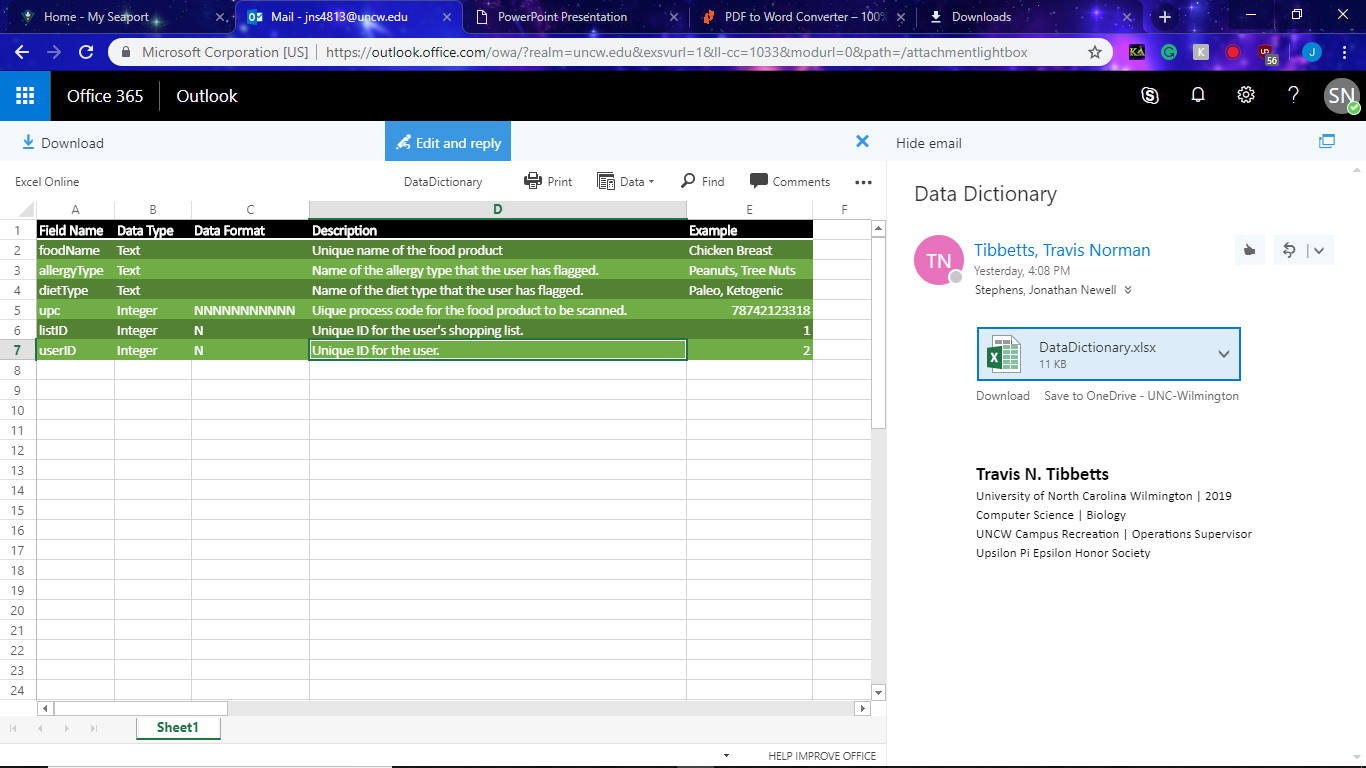
In the fully completed and ideal version, Grocer would be able to also show nutritional information on items. Showing calorie count, protein, carbs, sodium, etc. so that the user can see it, but also making comments on the whole of the cart. For example, if the cart has a lot of items that are high in calories, then grocery would mark the cart, and items, with a warning flag to show the user that this is something to be aware of. Grocer doesn’t aim to make someone be healthy, it more so is designed to let the user know exactly what they are consuming.







## **Data Dictionary**



## **Limitations and Constraints**

Currently, the limitations to Grocer consist of several issues with the database management, the operating system and platform, and the scope of development. The database is being developed with SQLite for ease of integration into Android Studio, but SQLite has several drawbacks compared to a full Database Management System, such as the lack of multivalued attributes. The database is being designed as a prototype system, because it is possibly limitless. The database contains different food products with unique Universal Product Codes (UPC) that vary from store and product name. Certain items such as fresh produce do not have UPCs to define the product, therefore each store has their own codes to sell the item. The database only contains items from grocery stores that are associated with a UPC. The database keeps track of the diet and allergy restrictions that each food product could possibly have. There may be unknown or not-tracked ingredients in the food that do not trigger an allergy or diet flag.

## **Non-functional Requirements**

### Platform Constraints

• The app is designed to be used on Android mobile devices.

• The app needs access to the device’s camera for the barcode scanner.

• The app will need space allocated on the devices hard drive.

### Modifiability

• The app will require roughly six months to achieve a deployable prototype stage.

• The user should be able to add their favorite products to the database with the app.

• Maintenance to the database and app will be an on-going effort.

### Portability

• Launching an iOS version of the app could potentially take over a year.

• With a small team, it could take roughly four person-months to develop the iOS version.

### Usability

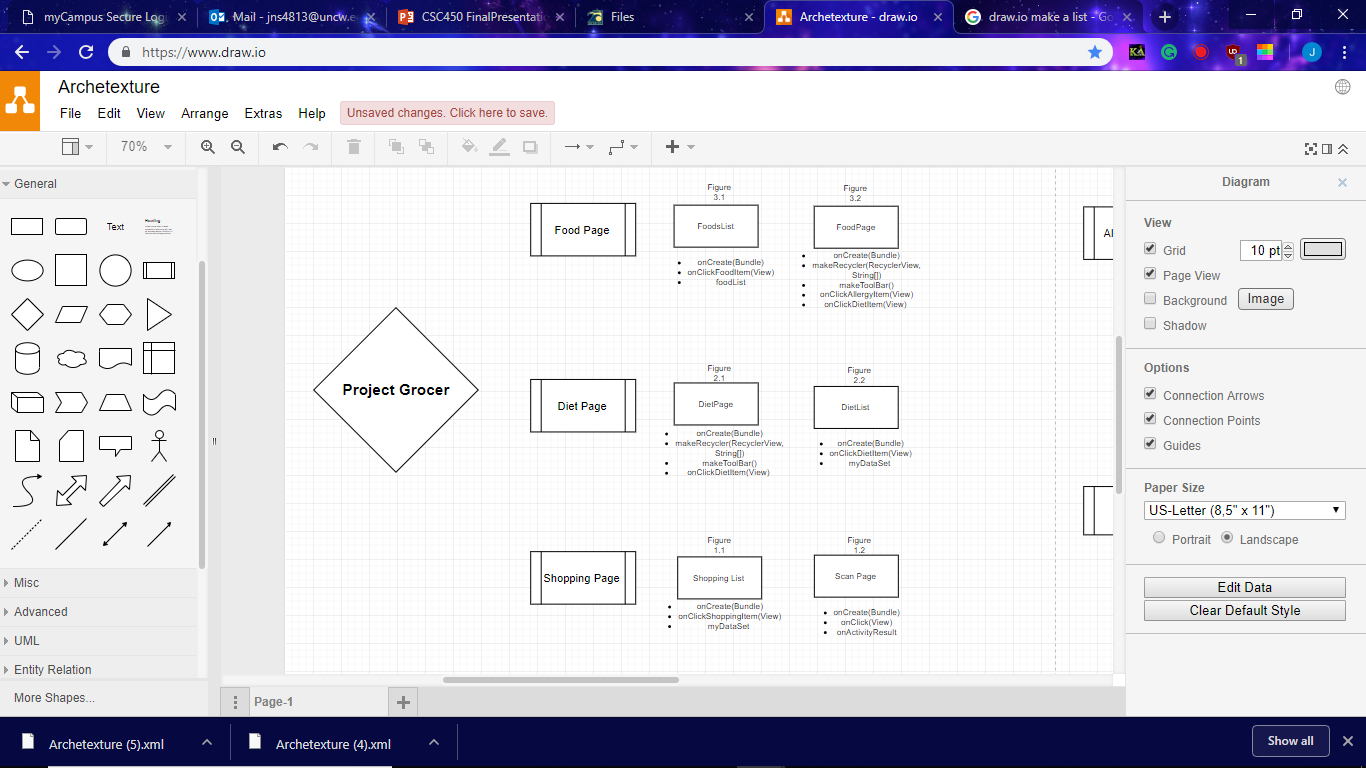
• The app should be designed to be easily operated.

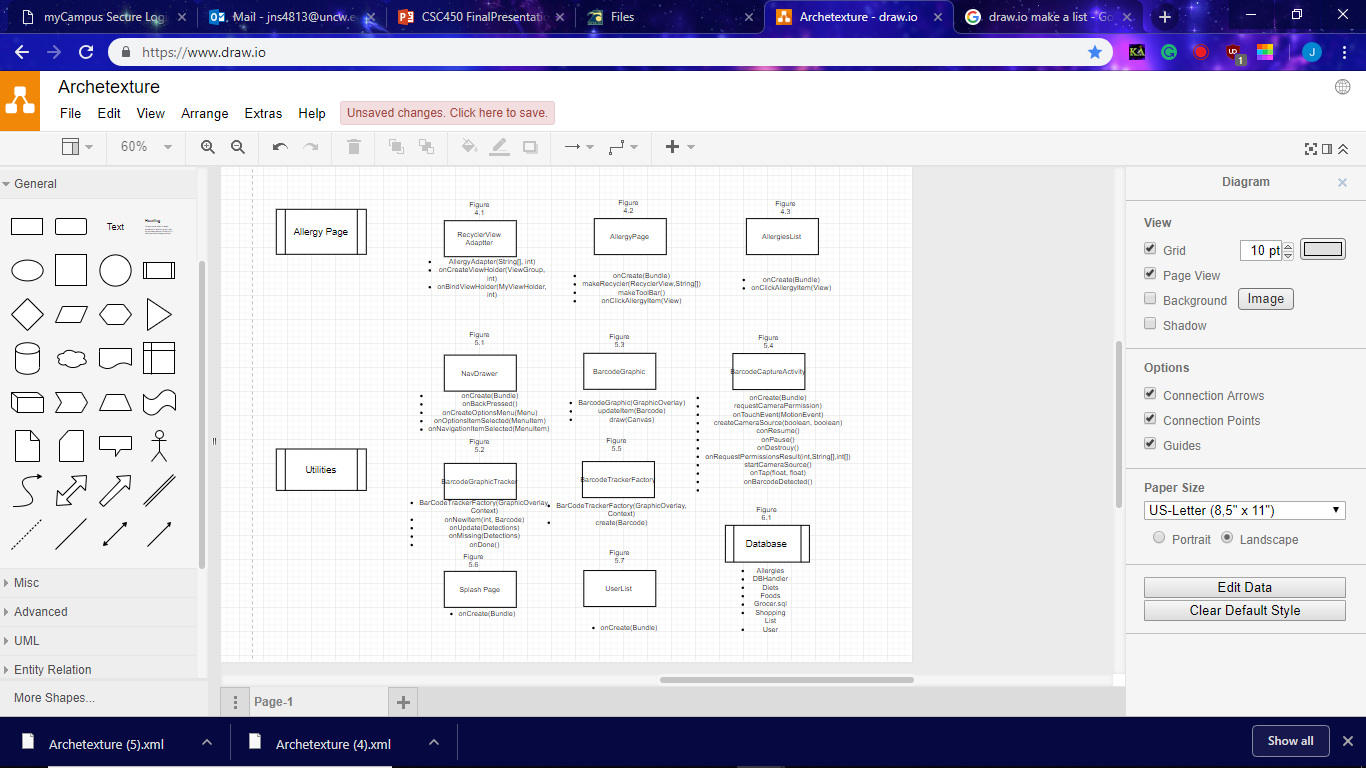
• The user should have an innate understanding of how to use the app.

• All of the functions of the app should be self-explanatory in their naming conventions.

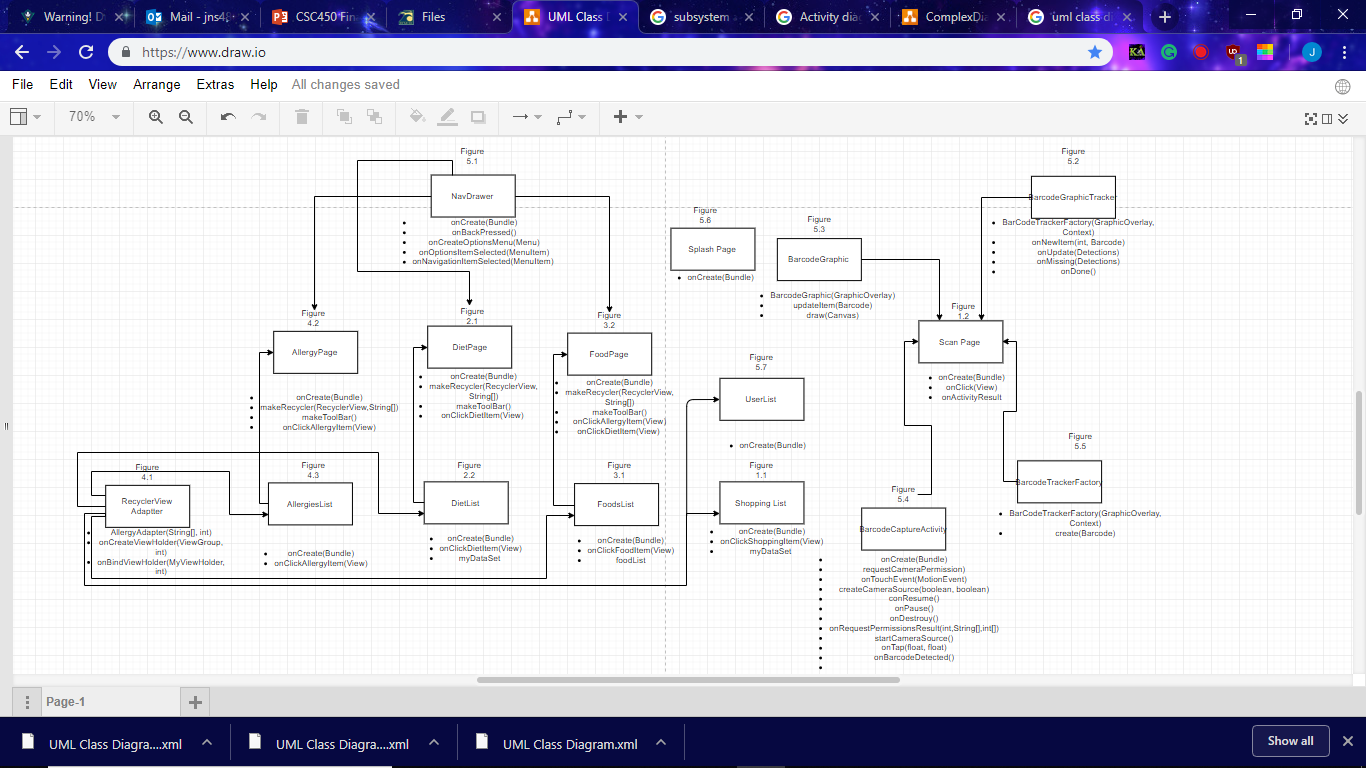
# Design Models

1. Design Models
   1. Graphic model of architectural design identifying subsystems.

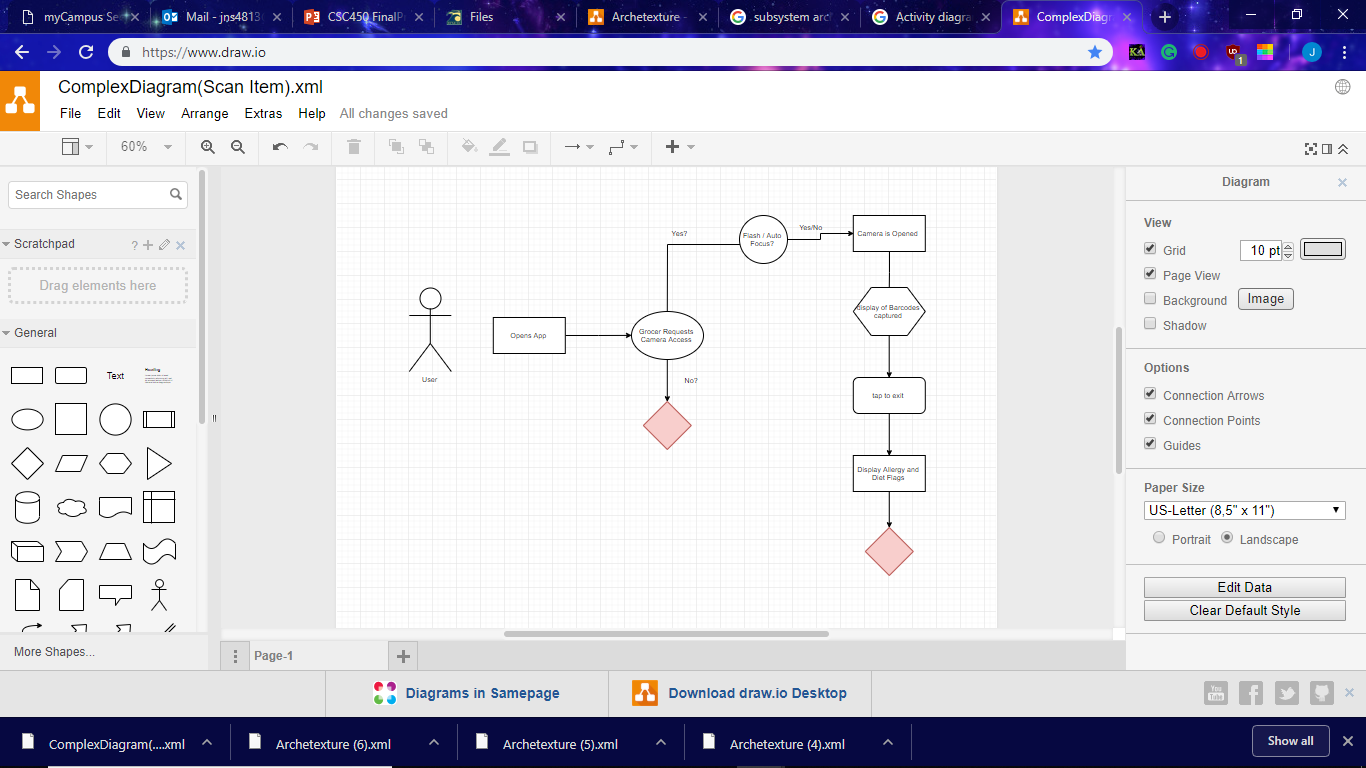




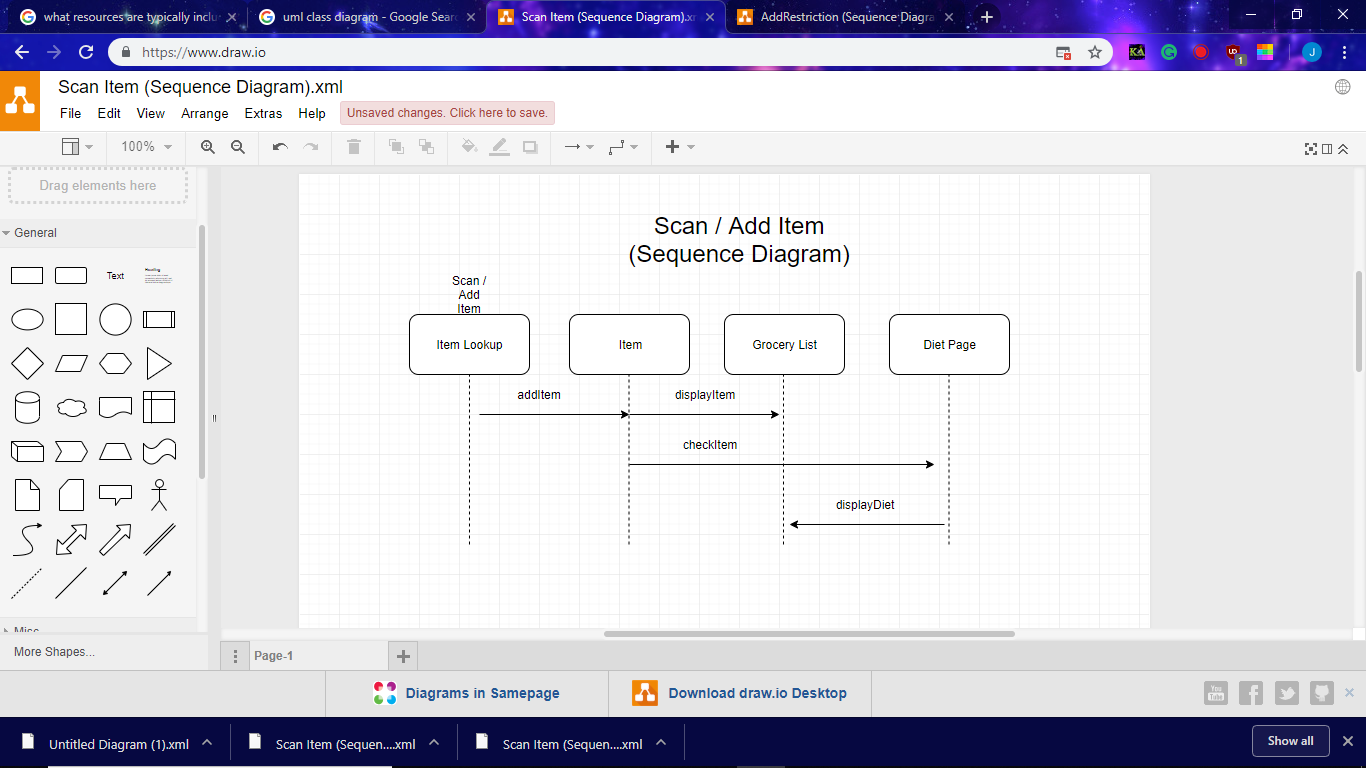
* 1. Textual Architecture Descriptions
     1. ***Figure 1.1 Shopping List***
        1. OnCreate(Bundle) : initializer
        2. onClickShoppingItem(View) : Event that takes place on clicking a shopping item
        3. myDataSet: Data set that the shopping list holds.
     2. ***Figure 1.2 Scan Page***
        1. onCreate(Bundle) : initializer
        2. onClick(View) : Event that takes place on Click
        3. onActivityResult : Event that takes place after an activity results.
     3. ***Figure 2.1 DietPage***
        1. onCreate(Bundle) : initalizer
        2. makeRecycler(RecyclerView,String[]): Container that holds the list
        3. makeToolBar(): creates Tool Bar
        4. onClickDietItem(View) : Event that takes place on clicking a Diet Item
     4. ***Figure 2.2 DietList***
        1. onCreate(Bundle) : initializer
        2. onClickDietItem(View) : Event that takes place on clicking a Diet Item
        3. myDataSet : Data set that the shopping list holds.
     5. ***Figure 3.1 FoodsList***
        1. onCreate(Bundle) : initializer
        2. onClickFoodItem(View) : Event that takes place on clicking a food item
        3. foodList : data set that contains all the food items.
     6. ***Figure 3.2 FoodPage***
        1. onCreate(Bundle) : initializer
        2. makeRecycler(RecyclcerView, String[]):Container that holds the list
        3. makeToolBar(): creates the tool bar
        4. onClickAllergyItem(View) : : Event that takes place on clicking an allergy item
        5. onClickDietItem(View) : : Event that takes place on clicking a diet item
     7. ***Figure 4.1 RecyclerView Adapter***
        1. AllergyAdapter(String[], int) : container that holds all the settings for the recyclers
        2. onCreateViewHolder(ViewGroup, int) : creates the view
        3. onBindViewHolder(MyViewHolder, int) : sets the text on the view.
     8. ***Figure 4.2 AllergyPage***
        1. onCreate(Bundle) : initializer
        2. makeRecycler(RecyclerView,String[]):Container that holds the list
        3. makeToolBar(): creates the tool bar
        4. onClickAllergyItem(View) : Event that takes place on clicking an allergy item
     9. ***Figure 4.3 AllergiesList***
        1. onCreate(Bundle) : initializer
        2. onClickAllergyItem(View) : Event that takes place on clicking an allergy item
     10. ***Figure 5.1 NavDrawer*** 
         1. onCreate(Bundle) : initializer
         2. onBackPressed(): controls the actions when back button is pressed.
         3. onCreateOptionsMenu(Menu) : creates options menu
         4. onOptionsItemSelected(MenuItem) : Event that takes place as an option item is selected
         5. onNavigationItemSelected(MenuItem) : Event that takes place as a navigation item is selected
     11. ***Figure 5.2 BarcodeGraphicTracker***
         1. BarCodeTrackerFactory(GraphicOverlay, Context) : whenever youre in a the camera view it tracks the barcode before the camera shot is taken
         2. onNewItem(int, Barcode) : Event that takes place as a new item is discovered.
         3. onUpdate(Detections) : event that takes place if the camera updates.
         4. onMissing(Detections) : event that takes place if no barcode is present
         5. onDone(): event that takes place as the barcode tracker finishes.
     12. ***Figure 5.3 BarcodeGraphic***
         1. BarcodeGraphic(GraphicOverlay) : Window that holds the barcode graphic
         2. updateItem(Barcode) : updates the graphic as the barcode is read
         3. draw(Canvas) : creates the graphic
     13. ***Figure 5.4 BarcodeCaptureActivity***
         1. onCreate(Bundle) : initializer
         2. requestCameraPermission() : requests camera permission from the phone
         3. onTouchEvent(MotionEvent) : what happens upon touching the screen
         4. createCameraSource(boolean, boolean) : creates where the camera source is coming from.
         5. onResume(): What happens when you resume the app.
         6. onPause():What happens when you pause the app.
         7. onDestroy():What happens when you terimnate the app.
         8. onRequestPermissionsResult(int,String[],int[]): what happens after the result of gaining permission
         9. startCameraSource(): starts up the camera source
         10. onTap(float, float) : what happens upon the screen being tapped
         11. onBarcodeDetected(): what happens upon the barcode being detected.
     14. ***Figure 5.5 BarcodeTrackerFactory***
         1. BarCodeTrackerFactory(GraphicOverlay, Context) : whenever youre in a the camera view it tracks the barcode before the camera shot is taken
         2. create(Barcode) : creates the Barcode Tracker
     15. ***Figure 5.6 Splash Page***
         1. onCreate(Bundle) : initializer
     16. ***Figure 5.7 UserList***
         1. onCreate(Bundle) : initializer
  2. Subsystem design
     1. UML class diagrams (with attributes and methods)



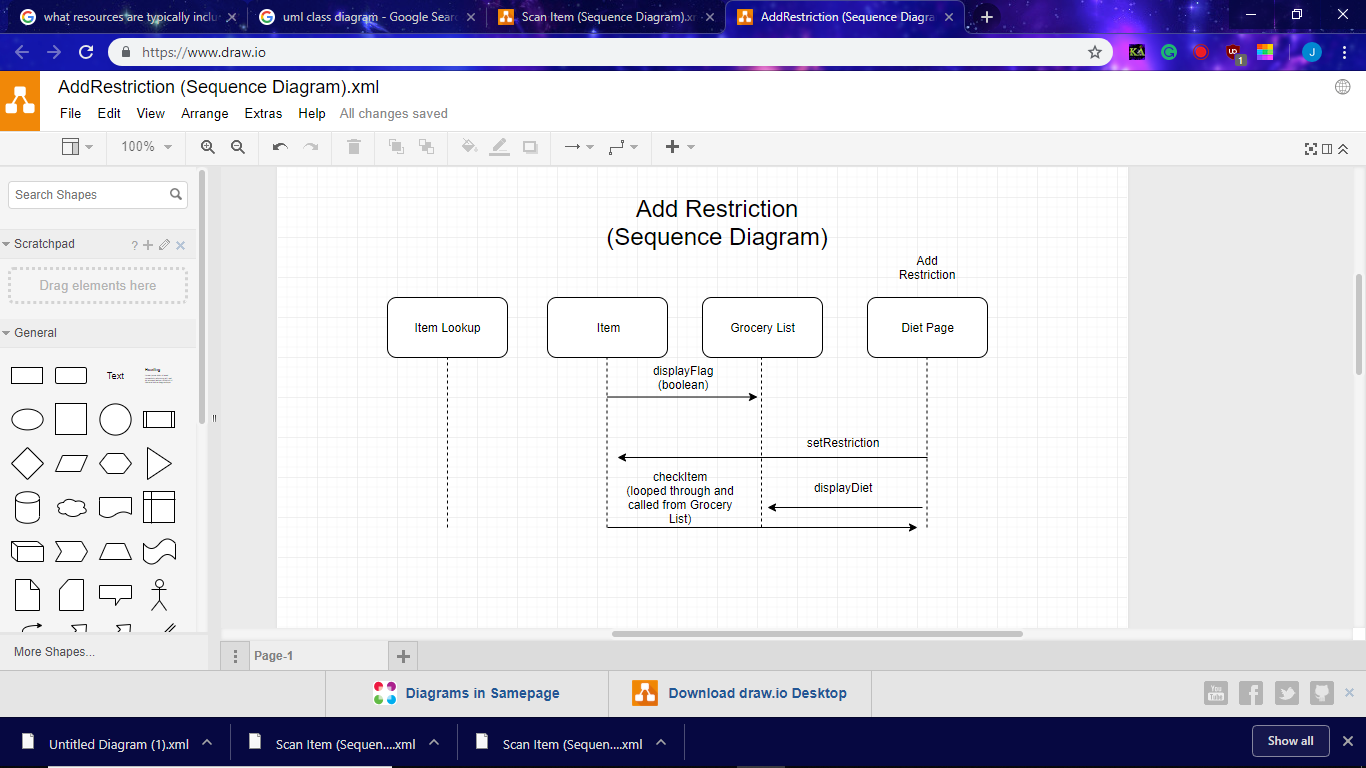
* + 1. Activity diagrams for complex tasks (that are not in Requirements)



* + 1. Sequence diagrams :
       1. Scan/Add Item



* + - 1. Add Restriction



* + 1. User interface design: I would like to see screenshots and design ideas and even hand drawn sketches. I would like to see your progress and your decision making.
    2. Grocer’s Design Pattern uses google's material design standards, which is the baseline for android development.
  1. Non-functional requirements:
     + **Platform Constraints** – Currently designing a prototype, when fully released it could be adapted to iOS as well.
     + **Modifiability** – Constantly addressed as the app is released, as far as database concerns. A growing database is a necessity of the app.
     + **Portability** – physical portability isn’t a concern, as it is a phone application, however virtual portability is dependent on iOS development which could take a year or more.
     + **Usability** – no concerns with usability, the app is fairly simple and navigation is straight forward
     + **Security**- Data will be localized so security isn’t an issue.

# Test Plan

a. Static Testing

i. Check style: Google’s material design standards

ii. Lines of code: 1400

x. Number of classes: 16

y. Number of methods: 58

Black and White Box Testing

* + 1. JUnit (or similar). How many JUnit tests were built? Regression testing?
       1. Android Studios Tester : 40
       2. No Regression Testing
    2. Code coverage of testing (White box)
  1. System Testing

i. Which use cases were tested and how did the tests go? Here you want a table that would have columns: Test Case #, Description, Intended Result, Actual Result, Completed by Whom

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Test Case # | Description | Intended Results | Actual Results | Completed by: |
| Scan Item | 1 | Using the barcode scanner to pull up diet page. | Scans Item and shows flags (if applicable) | Scan Item and show flags. | Jon Stephens |
| Add Item | 2 | Adding an item to the grocery list. | Adds item to Grocery list | Grocery list is a hardcoded list. | Jon Stephens |
| Add Restriction | 3 | Adding a restriction to the grocery list. | Adds Restriction to Grocery List | Grocery list is a hardcoded list.. | Jon Stephens |
| View Grocery List | 4 | Viewing the grocery List. | Opens Grocery List | Grocery list is a hardcoded list | Jon Stephens |
| Remove Item | 5 | Removing an Item from the grocery list. | Removes item from Grocery List | Grocery list is a hardcoded list. | Jon Stephens |
|  |  |  |  |  |  |

# Major Software Functions

* 1. Implemented -
  2. Not implemented -

# Problems Encountered

1. **Lack of Time** – With less time to work on this project than anticipated, hurricane, this project was already difficult to complete to the manner we would like, but accounting for the additional work piled on from other classes and having extracurricular activities, such as work, sports, and personal issues, it has delayed the rate of our progression on this project.
2. **SQLite** – Limitations – Resolved – SQLite’s inability to easily add queries made it difficult to categorize foods as needed but we were able to work around the problem.
3. **Bar Scanner** –Identifying the helpers and properly dealing with them, along with the unforeseen errors that would be thrown.
4. **Lack of Understanding** – For some of the software document, and for android development, we were not well versed in the subject entering into this class. Having to teach ourselves these subjects, while also being attentive to all of our other classes, slowed down the speed at which work was completed.

# Bibliography

“40 Use Case Templates & Examples (Word, PDF).” *Template Lab*, 16 Apr. 2018, templatelab.com/use-case-templates/.

*Food Composition Databases Show Foods List*, ndb.nal.usda.gov/ndb/search/list.

“ACAAI Public Website.” *ACAAI Public Website*, acaai.org/.

“Medical Information and Health Advice You Can Trust.” *Healthline*, Healthline Media, www.healthline.com/.

Salary.com. “Hourly Wage for Mobile Applications Developer.” *Salary.com*, www1.salary.com/Mobile-Applications-Developer-II-hourly-wages.html.