Got Eggplant?

CMPE 137

Team Swifters

Thinh Le, Cherie Sew, David Tran, and Timothy Wu

Table of Contents

Introduction and Motivation

Have you ever wanted something that could keep track of your favorite grocery items and put them onto a list that would find a location to buy them all? With the Got Eggplant mobile application, we decided to do exactly that.

This app has signup and login screens for authentication and saving account/user preferences through the use of Firebase. From the application, we have the ability to add items to the grocery list by scanning their barcodes. The grocery list is used to remind users which groceries to buy. The pantry list is used to display items in the household. The application can search to find the nearest location to buy grocery items. Once purchased, the user can check off the item in order to add the item to the pantry list, where they can check if they have the item or not.

High Level Design

From a high level point-of-view, the high level design consists of multiple views in the storyboard where each view is controlled by a view controller. All view controllers for views are subclasses of the UIViewController class. This view controller acts as the controller for the model, which consists of three Swift files. One is the barcode reader, which imports the AVFoundation in order to access camera and scan barcodes of any format. The next one is a location manager that imports both CoreLocation and MapKit to grab the location of the user and find the nearest location to buy grocery items. Lastly, is the grocery and pantry lists, that allows the user to keep track of items that need to be purchased, or is already in the household.

All view controllers are properly segued (linked to tab bar) and held as children to the UINavigationController. This allows us to create many separate view controllers, which each have their own data file.

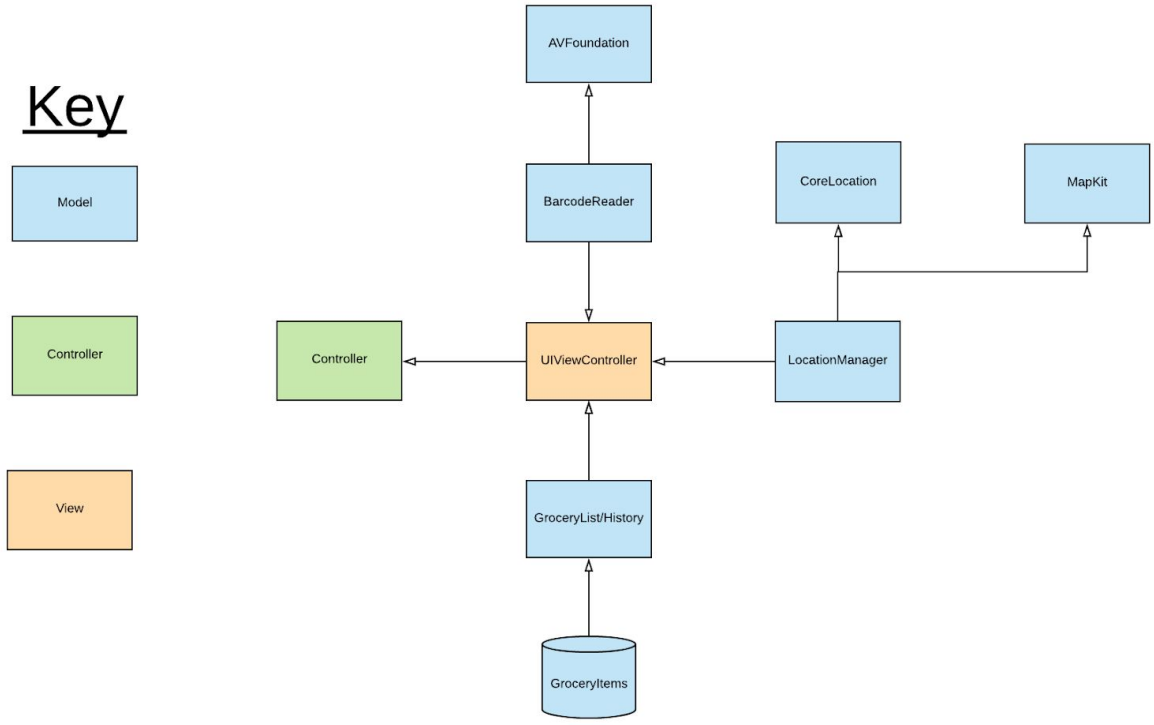


Figure 1. Architecture diagram for the Got Eggplant? App

Component Level Design

To go into each component more in detail, let's begin with the authentication part of the app. On app startup, the launch screen shows the logo of Got Eggplant? app with two buttons for signup and login. This launch page segues into a view controller for authentication, for the signup page and the login page. Only difference between the two views is that the signup page has an additional field for the username.

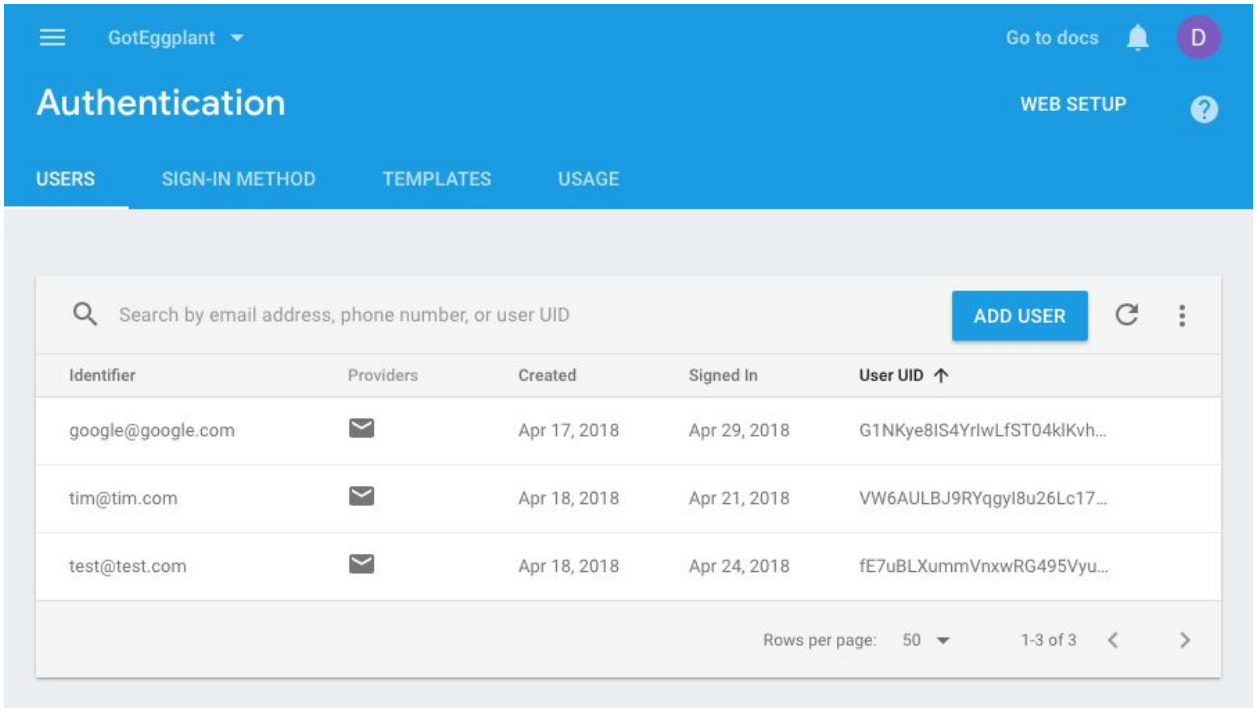


Figure 2. Authentication system through Firebase

Once authentication is verified, the rest of the application has a navigation bar to access each feature and its view, which leads to the grocery list, pantry list, store locator, and barcode scanner.

The grocery and pantry lists both utilize the Firebase Realtime Database by storing the items and its timestamp in a DB. This is done using the podfile and firebase files that connect

from the Xcode project to Firebase's cloud NoSQL database. Also, each list has a model file that mostly focuses on utilizing UITableViews to output item class values held in a 2D array.

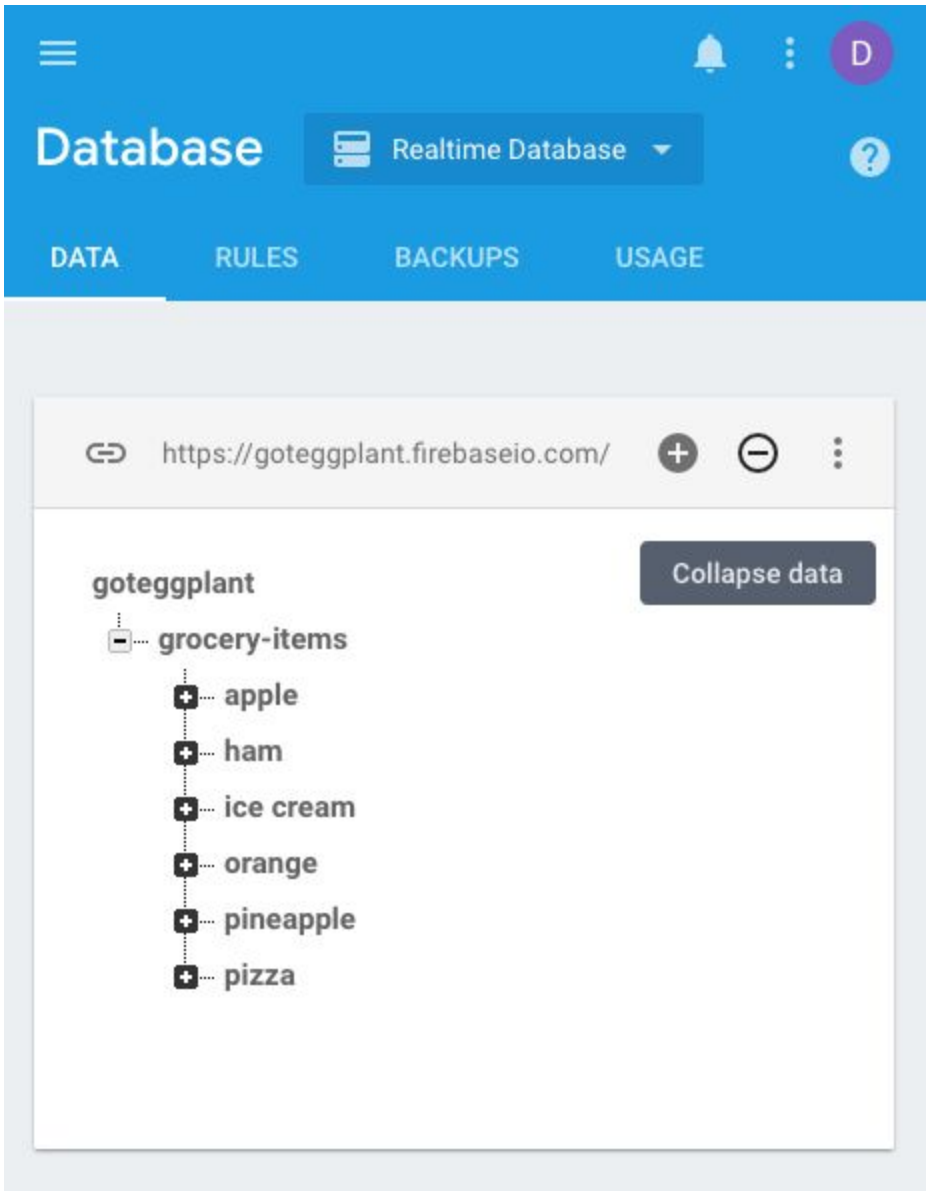


Figure 3. Grocery list table of test user

The barcode scanner utilizes the back camera from the iOS device to scan a barcode. When scanned, it will output the information onto the screen. From there, the user can add the data to Firebase, where it will be added to the grocery list of the app. This is done with the code snippet below.

```
14
15    var captureSession : AVCaptureSession?
16    var videoPreviewLayer: AVCaptureVideoPreviewLayer?
17    var barcodeFrameView: UIView?
18
19    private let supportedCodeTypes = [AVMetadataObject.ObjectType.upce,
20                                      AVMetadataObject.ObjectType.code39,
21                                      AVMetadataObject.ObjectType.code39Mod43,
22                                      AVMetadataObject.ObjectType.code93,
23                                      AVMetadataObject.ObjectType.code128,
24                                      AVMetadataObject.ObjectType.ean8,
25                                      AVMetadataObject.ObjectType.ean13,
26                                      AVMetadataObject.ObjectType.aztec,
27                                      AVMetadataObject.ObjectType.pdf417,
28                                      AVMetadataObject.ObjectType.itf14,
29                                      AVMetadataObject.ObjectType.dataMatrix,
30                                      AVMetadataObject.ObjectType.interleaved2of5,
31                                      AVMetadataObject.ObjectType.qr]
32
33    override func viewDidLoad() {
34        super.viewDidLoad()
35        let deviceDiscoverySession = AVCaptureDevice.DiscoverySession(deviceTypes:
              [.builtInDualCamera], mediaType: AVMediaType.video, position: .back)
36        guard let captureDevice = deviceDiscoverySession.devices.first else {
37            print("Failed to access the camera.")
38            return
39        }
40        do {
41            let input = try AVCaptureDeviceInput(device: captureDevice)
42            captureSession?.addInput(input)
43
44            let captureMetadataOutput = AVCaptureMetadataOutput()
45            captureSession?.addOutput(captureMetadataOutput)
46
47            captureMetadataOutput.setMetadataObjectsDelegate(self, queue: DispatchQueue.main)
48            captureMetadataOutput.metadataObjectTypes = supportedCodeTypes
```

Figure 4. Code that updates data detected in scanned barcode

Technology Choices

Our primary motivation behind all of our technology choices was ease of implementation. We chose Firebase, MapKit, CoreGraphics, and AVFoundation due to the fact that those would be the easiest to deal with when implementing the various features of our app.

Firebase is by far the easiest way to implement authentication, data storage, and analytics information for mobile applications. For our specific app, we only required authentication and a database for implementation. Firebase allowed us to spend minimal time worrying about which DB to use and how to implement email and password storage from user information. Because of this, Firebase was instrumental in our success for building the login and logout of our app.

The decision to use the MapKit API was the easy integration to our iOS app with the detailed layouts, annotations, location management, and querying. The app on start up displays the visible area in the view, which is set by latitude and longitude. The app requests always allowing access to ensure a level of authorization only when using the app because it offers a genuine benefit to user privacy. The developer based tools used from Map Kit were the Core Location Manager tools, the Map View Delegate, the Marker Annotation Views. Utilization of the kit allowed for smooth map interface along with functionality of user interaction in the map view.

AVFoundation is absolutely necessary to even get access to the camera and apply functionality to the barcode scanner. However, the protocol AVCaptureMetadataOutput is what allows our app to store the barcode information, which can be stored in our Firebase DB. For this reason, AVFoundation was vital to all camera-related functionality and AVCaptureMetadataOutput was responsible for storing the barcode data.

Description of Features with Final Screenshots

Note: Adobe XD was used for a majority of the static page views.

Main - Users have a main page to redirect to creating an account or logging in. Two buttons were created for each respective feature.

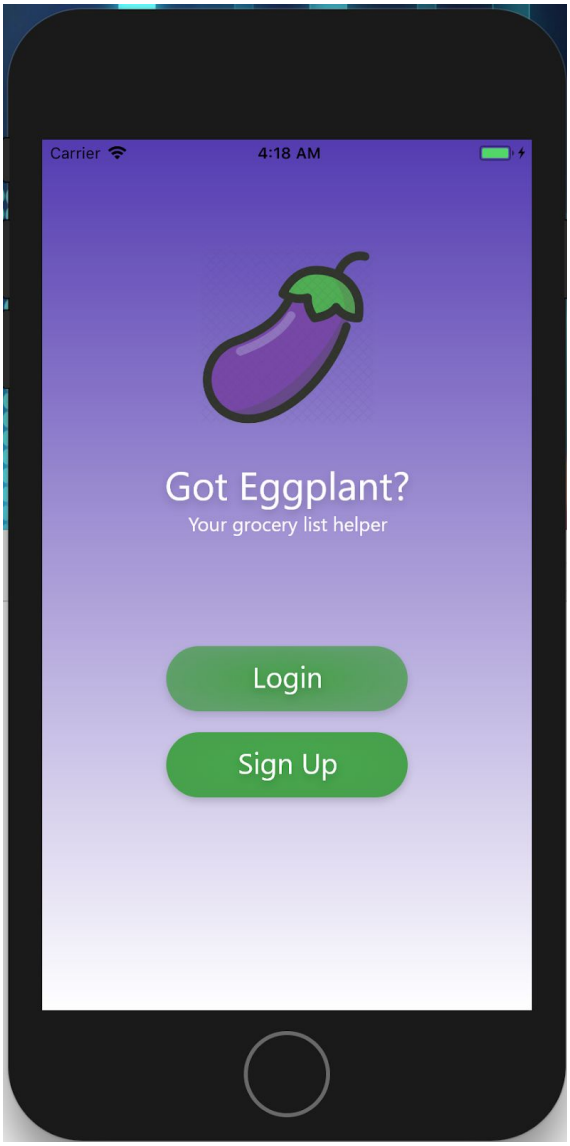


Figure 5. Main Page

Signup - Users can create an account with a username, email, and password. Three textfields were created for each respective feature. A button was created for “sign up”. Users also have the option of going back to the main page.

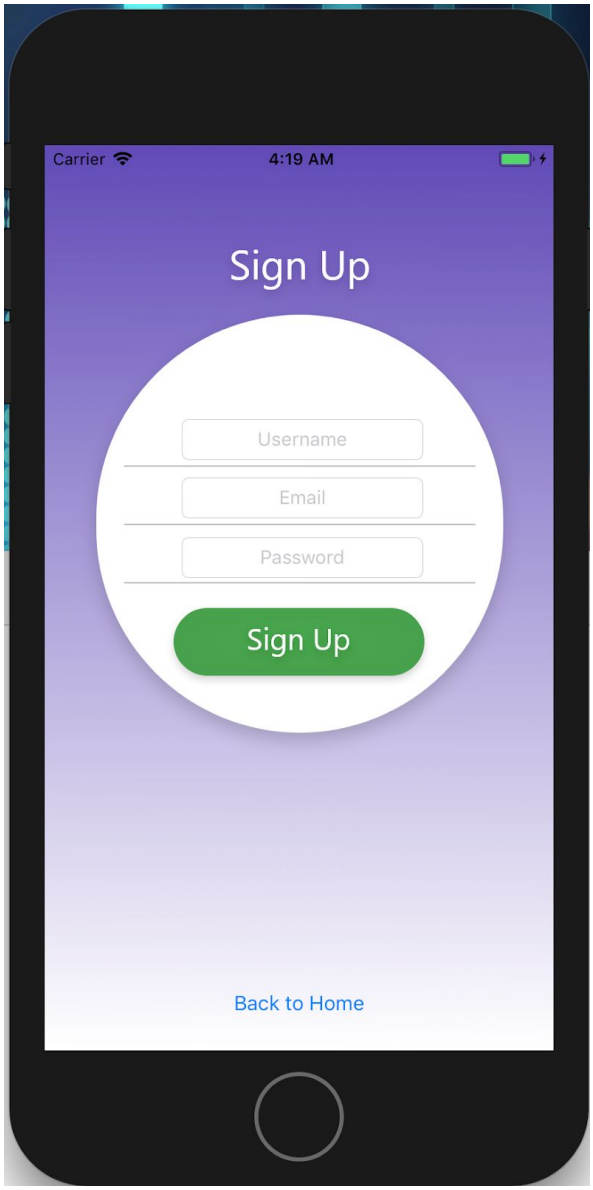


Figure 6. Signup Page

Login / Logout - Users can access their account after they have signed up for one. Text fields for email and password were created, in addition to the “login” button. Users also have the option of going back to the main page. On the settings page, users can log out of their account.

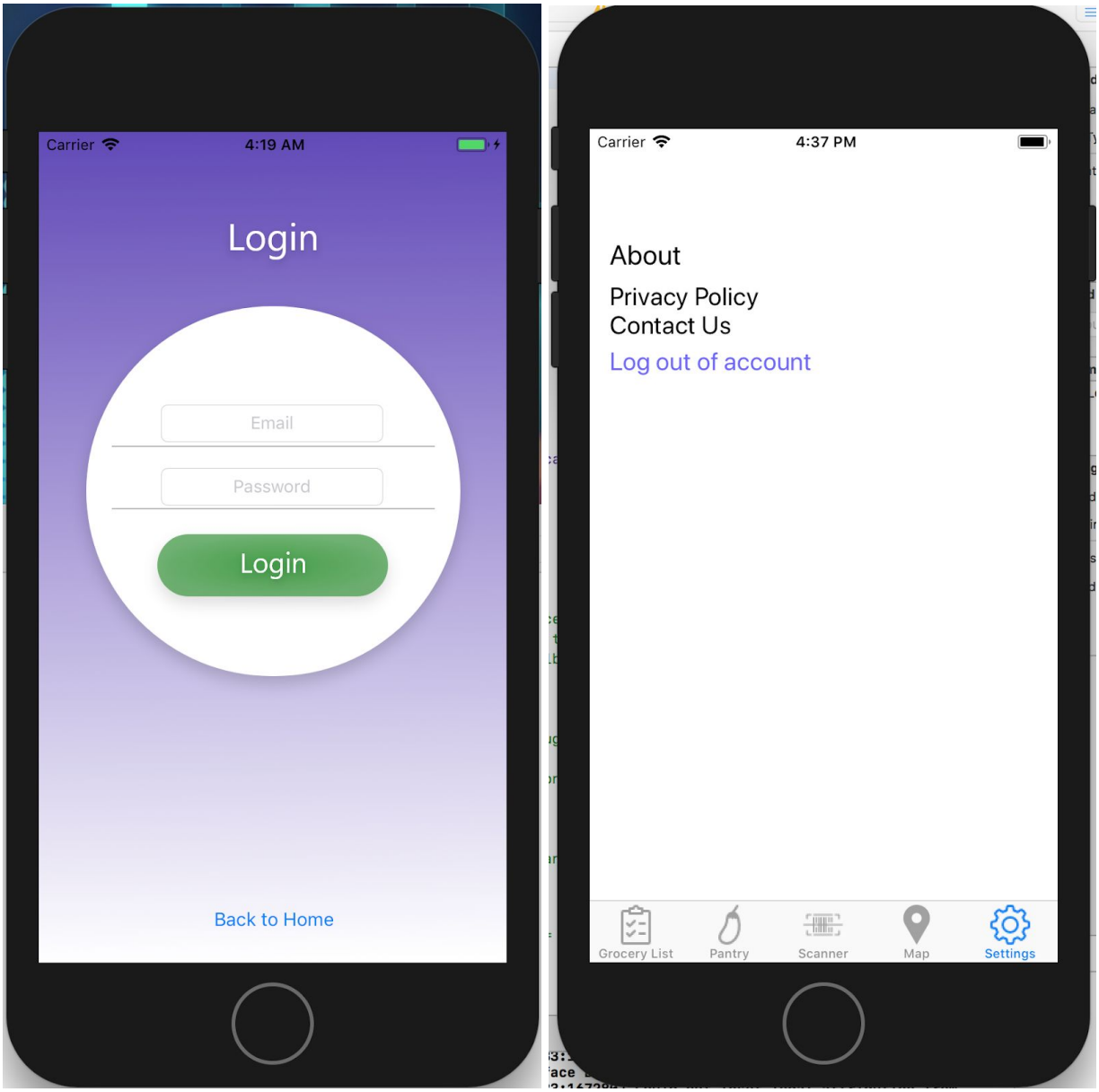
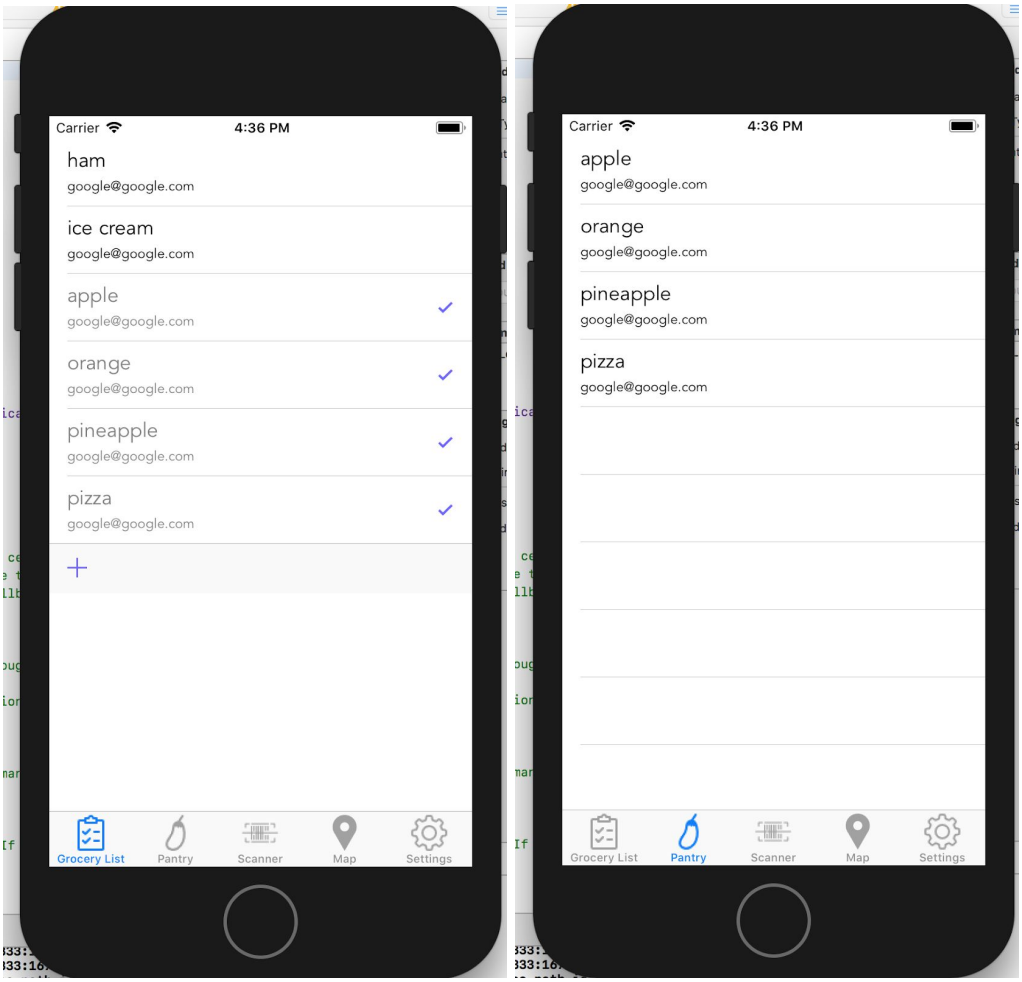


Figure 7. Login / Logout Pages

Grocery / Pantry Lists - Users can create, update, and delete grocery items. Users can check off individual items in a grocery list. The pantry list shows only the “completed” items from the grocery list. This all sits on top of a tableview and is updated from a live aspect from the Firebase database.



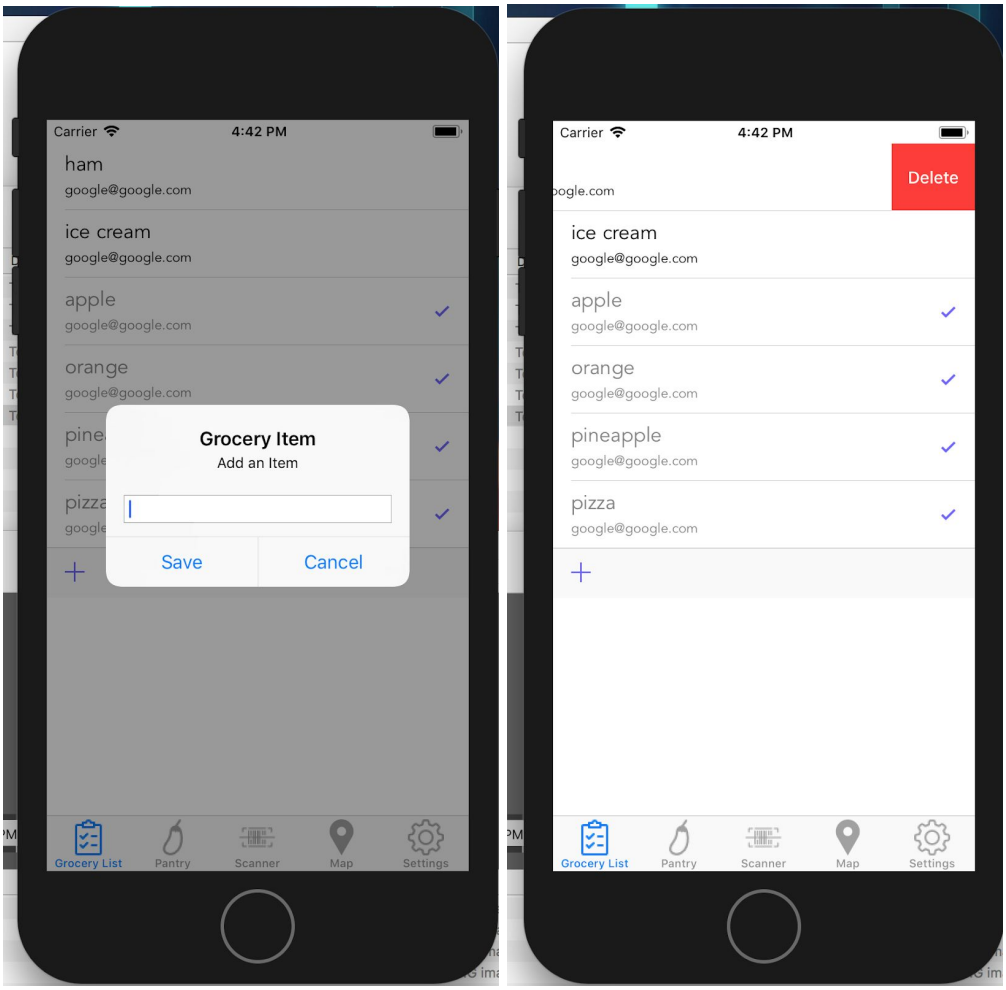
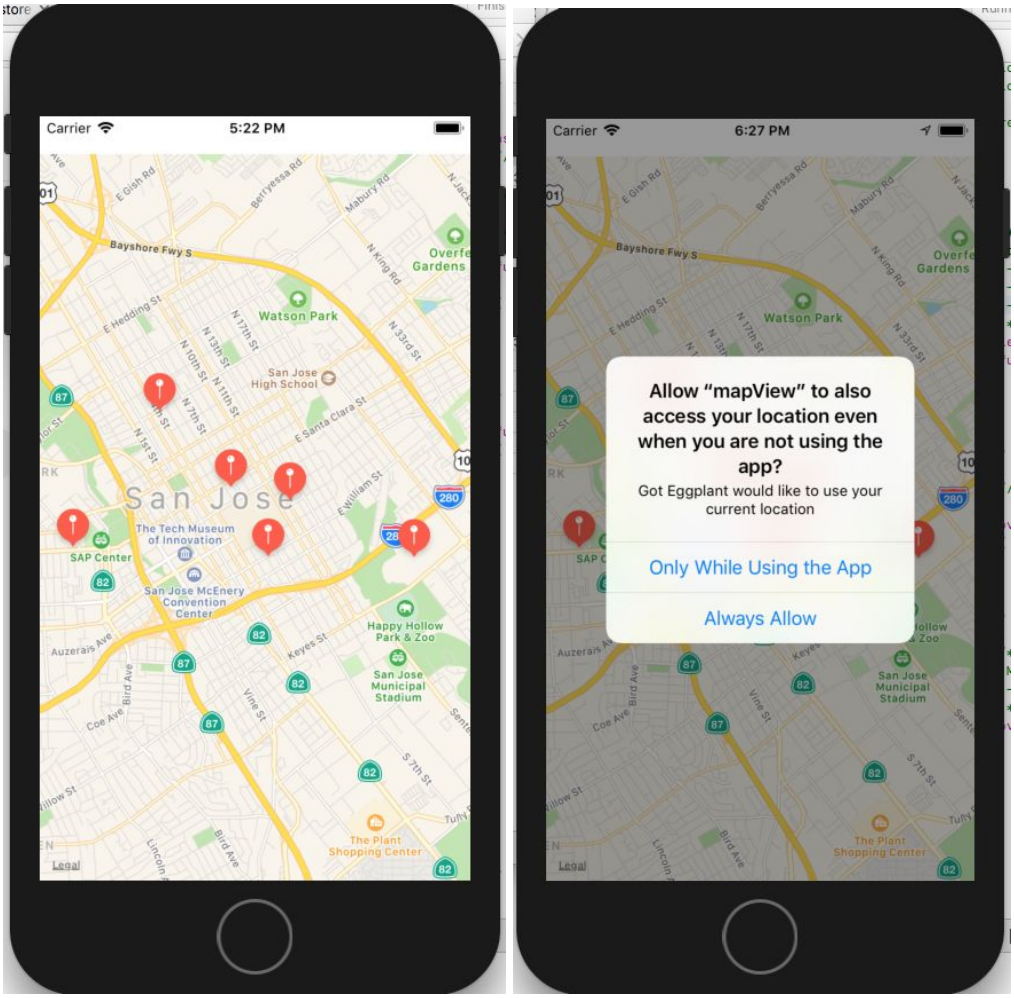


Figure 8. Grocery / Pantry Lists Pages

Geolocation of Grocery Stores - Users can browse their favorite shopping destinations or discover new ones through the discipline filter option from the app. From the intuitive markers, users can easily tap and view the location names and available inventory. The current user location activation feature allows for convenience of navigation to their destination in minimal time and distance.



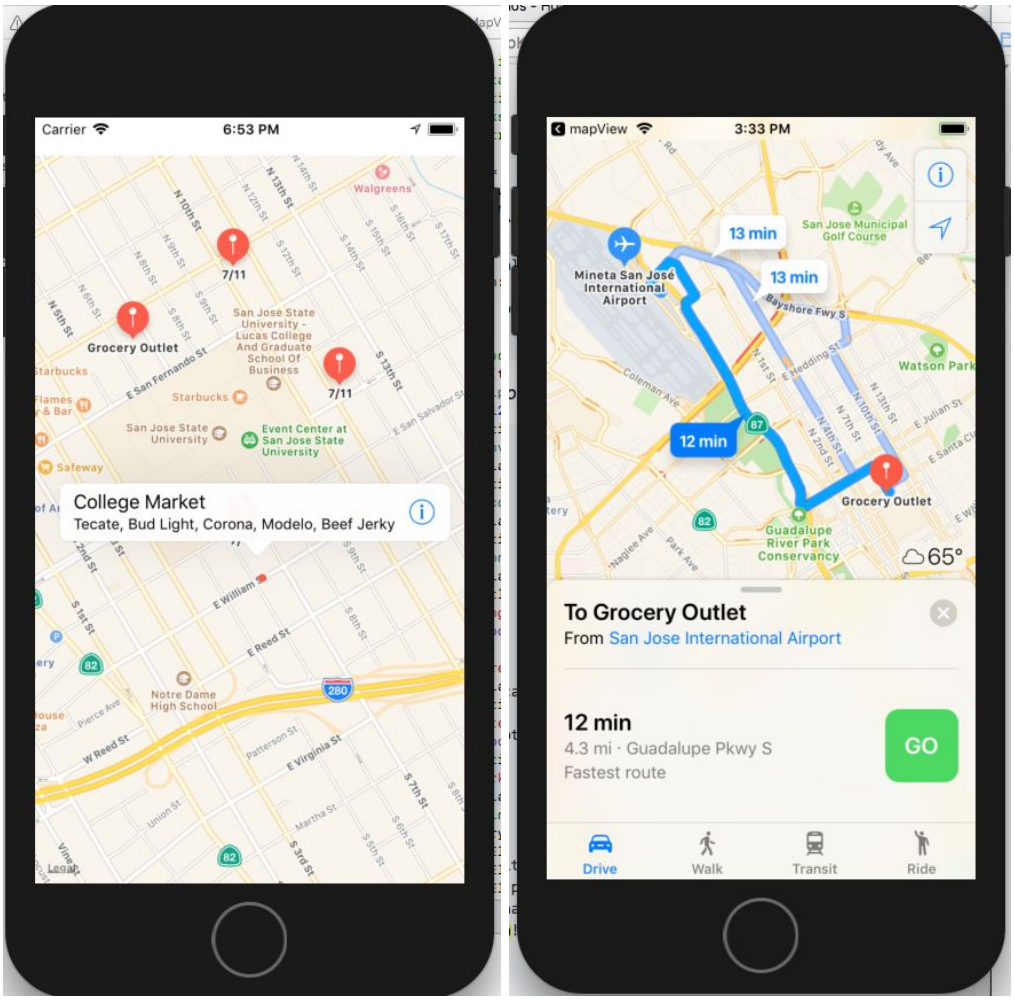


Figure 9. Map Pages

Barcode Scanning - Users can scan the barcode of an item using their iPhone camera. The iOS application will then update the item information accordingly. If barcode scanning is unavailable, user can manually input the data for the items from the grocery list screen.

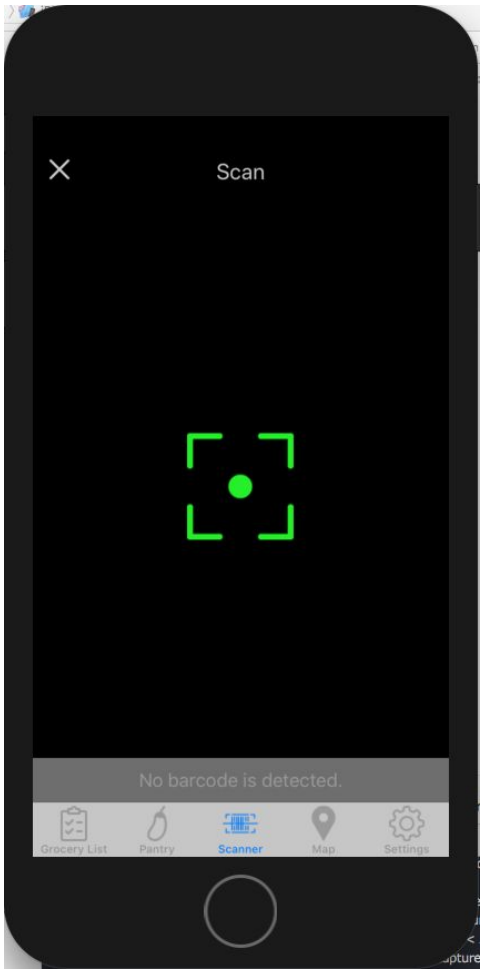


Figure 10. Barcode Scanning Page

Navigation Bar - A navigation bar that connects to each individual page.

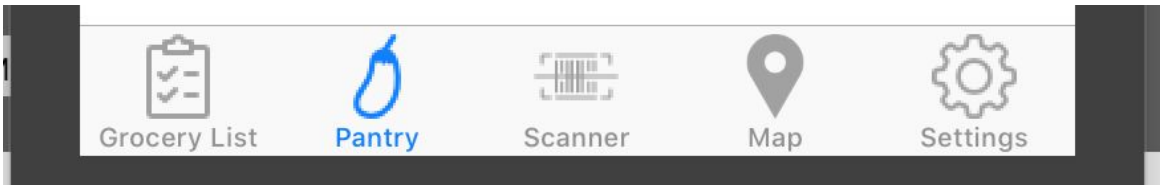


Figure 11. Navigation Bar

Testing Plan Executed and Results

We tested Got Eggplant using several methods. As mentioned in class, we added print statements to functions in our app to make sure that the feature is working as expected. For example, we added print statements for our signup and login features that will print if the user has successfully signed up or logged in through the console.

We also tested Got Eggplant by uploading the app onto an iPhone. The first feature to test was the signup page. If the user tries to signup without an email or password, an error will inform the user to input email and password. The error shown is similar for login. We also tested the grocery and pantry lists features. When we add an item to the grocery list, the item will be saved to our Firebase database in live view. We also confirmed that when the item in the grocery list has been checked off, it appears in the pantry list.

The map view test consisted of browsing the locations indicated within loading the view and iterating through individual markers in the annotation array. The locations were tapped and configured for direction calculating and reporting feedback. The correct addresses were mapped out and directed out when compared to a working map featured app e.g. Google Maps.

We tested the barcode scanner when we uploaded the app to an iPhone. We tried to scan the barcode of a grocery item and the results were that the information stored in the barcode is stored to Firebase. The results were then updated to the grocery list model, which shows data in the grocery list view. Overall the test results were satisfactory and the application worked as expected. The user can add items in the grocery lists with no problem and checking off the items from the grocery list will allow the same item to appear in the pantry list.

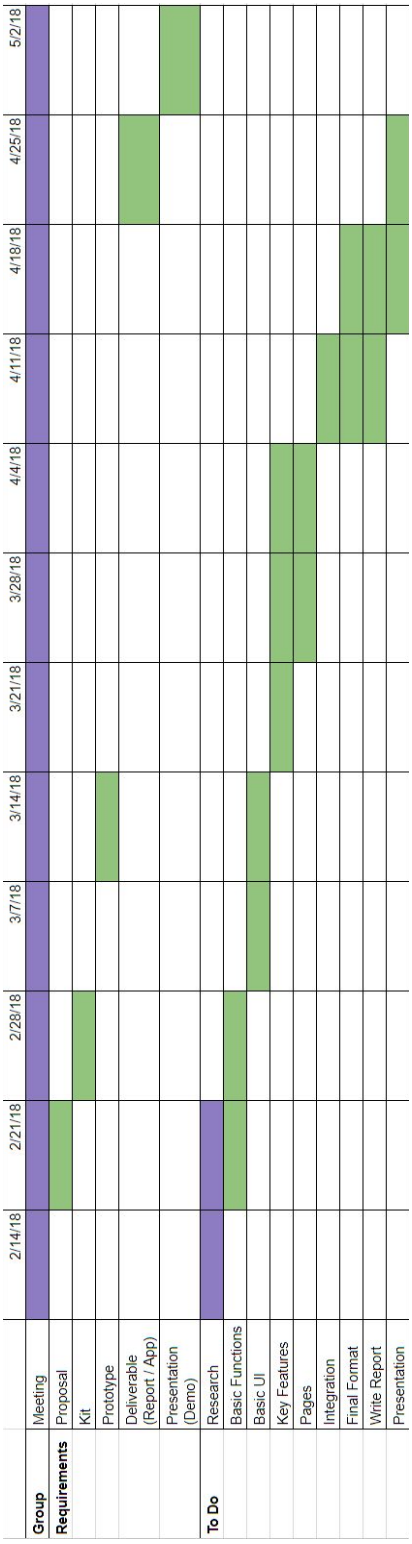


Figure 12. Progress Timeline

Lessons Learned and Possible Future Work

There are a lot of things that we could have done to have this project run more smoothly. We should have quickly adopted Firebase over Cloudkit due to the many issues that arose from Cloudkit authentication. Firebase ended up being the ideal database to store user authentication as well as an ideal database to store both barcode data as well as grocery list information.

Also, we ended up behind schedule on the the project after turning in the login/logout portion of the project, but that is more due to people's schedules and the nature of the class. Every class has their project due at about the same time, so we could have possibly started on the project very early (like around early March).

For the system design, we were not clear about how we wanted to connect all of the views. It took a while for us to settle on connecting all of the view controllers with a navigation controller instead of segues for all of the views. Had we settled on a proper system design from the start, we would have most likely had very few issues with source control.

Additionally, we wanted to add features that require the use of a calendar. However, a lot of tutorials we found online were only for the existing Apple Calendar. It is possible to add the expiration dates of food items to the Apple calendar but it wasn't ideal because it would have clogged up the user's Apple Calendar.

Some possible future work includes adding expiration date notification and calendar views. As mentioned, calendar views was not easy to implement because we would have to call data from Firebase and build a calendar view from scratch. However, given more time, it would have been possible to build a calendar view, implement Event Kit, and provide notifications to inform the user when an item is expiring.