Table of Contents

1. **Introduction and Background** 3
   1.1. **Statement of Problem Area** 3
   1.2. **Background** 3
   1.3. **Brief Project Description** 3
   1.4. **Purpose/Objectives/Justification of Projection** 3
   1.5. **Team work Assignment and Accomplished** 3
2. **System Functional Specification** 3
   2.1. **Functions Performed** 3
   2.2. **User Interface Design** 3
   2.3. **User Input Preview** 4
   2.4. **User Output Preview** 4
   2.5. **System Data Base / File Structure Preview** 4
   2.6. **External and Internal Limitation** 4
   2.7. **User Interface Specification** 4
      2.7.1. User Interface Design 4
      2.7.2. User Screens/Dialog 5
      2.7.3. Online Help Material 5
      2.7.4. Error Conditions and System Messages 5
3. **System Performance Requirements** 5
   3.1. **Efficiency** 5
      3.1.1. Table View Controller cell reuse 5
      3.1.2. Parse query save/fetch in background 5
      3.1.3. iOS use of ARC 5
   3.2. **Reliability** 6
      3.2.1. Description of Reliability Measures 6
      3.2.2. Error/Failure Detection and Recovery 6
      3.2.3. Allowable/Acceptable Error/Failure Rate 6
   3.3. **Security** 7
      3.3.1. Data Security 7
4. **System Design Overview** 7
   4.1. **iOS Project Class Diagram** 7
   4.2. **Description of System Operation** 9
   4.3. **Implementation Languages** 9
5. **System Data Structure Specifications** 9
   5.1. **Other User Input Specifications** 9
      5.1.1. Identification of Input Data 9
      5.1.2. Source of Input Data 9
      5.1.3. Input Medium and/or Device 9

5.1.4. Data Format/Syntax 9
5.2. **Other User Output Specifications** **9**
5.2.1. Identification of Output Data 9
5.3. **System Database/File Structure Specification** **10**
5.3.1. Modules Accessing Structures 11

6. **Module Design Specifications** **12**
6.1. **Module Functional Specification** **12**
6.1.1. Functions Performed 12
6.1.2. Module Interface Specifications 12
6.1.3. Module Limitations and Restrictions 13
6.2. **Module Operational Specification** **13**
6.2.1. Locally Declared Data Specifications 13
6.2.2. Storyboard Specification 14
6.2.3. Description of Module Operation 14
6.2.4. Locally Declared Data Specifications 15
6.2.5. Algorithm Specification 15

7. **System Verification** **16**
7.1. **Items/Functions to be Tested** **16**
7.2. **Justification of Test Cases** **16**
7.3. **Test Run Procedures and Results** **16**
7.4. **Discussion of Test Results** **16**
7.5. **Test cases** **16**

8. **Conclusions** **26**
8.1. **Summary** **26**
8.2. **Problems Encountered and Solved** **26**
8.3. **Suggestions for Better Approaches to Problem/Project** **26**
8.4. **Suggestions for Future Extensions to Project** **26**

9. **User Manual** **26**

Introduction and Background

Statement of Problem Area

Students on campus have difficulty finding one another. The reasons include: not knowing the names of buildings, not knowing the layout of the campus, and forgetting to remind each other of scheduled meetings.

Background

Current solutions to this problem require traditional communication methods such as sending phone calls and text messages. Most of these are obtrusive to the classroom and are unreliable. For college campuses, students are often unfamiliar with the new and large environments so even directories are required to identify locations.

Brief Project Description

CSULB FF is an easily accessible smart phone app that solves this problem by providing a quick way to locate friends on the campus as well as providing a helpful way to navigate towards them. The navigation is implemented with both a map to locate all the user’s friends and an augmented reality display to guide users towards them. It allows for registration via the popular social networking service Facebook to quickly connect the users to their friends.

Purpose/Objectives/Justification of Projection

CSULB students will become better connected with others by always knowing where others are. Users will be able to quickly meet up with one another, allowing students to save time scheduling and discussing their locations. New students who use the app will become better acquainted with the campus.

Team work Assignment and Accomplished

CSULB FF was developed using agile processes. Every week all members gathered to collaborate on the project. Each sprint focused on developing a set of user stories. At the end of the semester our app was completed finishing all set user stories except for implementing the use of schedules. In total, about 700 person-hours were spent in app development, resulting in about 3000 lines of code.

System Functional Specification

Functions Performed

CSULB Friend Finder is an iPhone app with three main functions: 1) viewing one’s friends on a map of the CSULB campus, 2) viewing one’s friends and their building locations on a textual friends list, and 3) updating one’s status and location on one’s own profile. These three functions, along with a fourth Settings tab, make up the four tabs of the iPhone app.

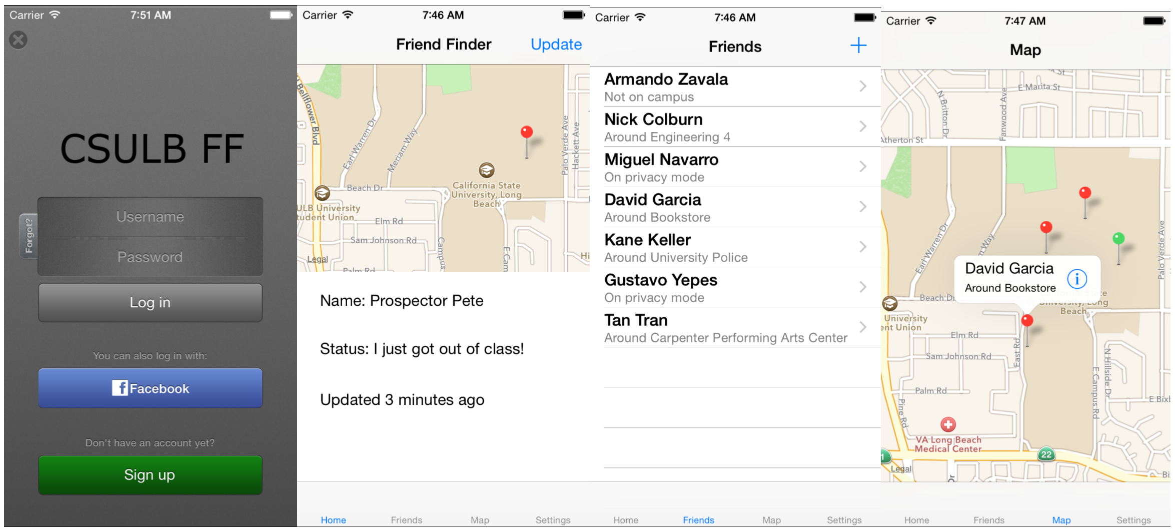
User Interface Design

The app is a tabbed iOS application. The four tabs are those mentioned above. The default tab is the tab for one’s profile.

User Input Preview
The signup process takes the new user's first name, last name, and email as inputs. Additionally, if the user is signing up using Facebook, a special Facebook session token is sent to Parse upon Facebook login and associated with the user in the CSULB FF database. Otherwise, for non-Facebook signups, a username and password is also input.

Updating one's status requires the entry of a 140-character status text. Upon update, the current geolocation will also be updated. Since CSULB FF is solely a location tracking service, rather than a social networking service, the status and current location cannot be updated separately.

User Output Preview
Output is displayed throughout the iOS app in the form of screens. Screenshots follow for previewing:



Name: Prospector Pete

Status: I just got out of class!

Updated 3 minutes ago

Armando Zavala
Not on campus

Nick Colburn
Around Engineering 4

Miguel Navarro
On privacy mode

David Garcia
Around Bookstore

Kane Keller
Around University Police

Gustavo Yepes
On privacy mode

Tan Tran
Around Carpenter Performing Arts Center

System Data Base / File Structure Preview
CSULB FF stores data on the databases and servers of a framework called Parse. The Parse database is a key-value store with table support. The tables used in CSULB FF are: User, Building, FriendRequest, and Friendship, which have self-explanatory names. Explanation follows in the section titled "System Data Structure Specifications".

External and Internal Limitation
The tabbed user interface of CSULB FF will be intuitive to new users because the four main functions (viewing one's profile, viewing the friends list, viewing the map, and changing settings) remain visible during the usas and Restrictions
Usage of CSULB FF requires internet and geolocation capability.

User Interface Specification

User Interface Design
The user interface is designed to be intuitive. One navigates through the four main tabs by clicking on each tab. There are also secondary tabs. For example, the AddFriend view is secondary to the Friends List view, while the UpdateStatus view is secondary to the Self Profile

view. One exits the former by pressing “Back”, while one exits the latter by pressing “Back” or “Update”. This is an example of how the interface was designed with simplicity in mind.

User Screens/Dialog

There is one login screen. If one is registering as a new CSULB FF user without Facebook, one goes through one screen to enter their First and Last name. If one is registering with Facebook, one goes through two screens: one to enter First and Last Name, and another to add recommended friends (one’s Facebook friends who also use CSULB FF).

The four tabs mentioned earlier in this section comprise the remaining screens.

Online Help Material

A user manual is available on the CSULB FF website.

Error Conditions and System Messages

Errors such as the failure of a pull or push from the backend server are handled by the display of an iOS alert, which tells the user that an error occurs and advises them to try again later. If the step was crucial, such as entering one’s info during signup, the app stalls at that view, so that the user may open the app again later and try submitting the required info again. Otherwise, the alert is displayed, and nothing else is done. Then, the user has the option to “Go Back” from the view, do other things within the app, and try again later.

System Performance Requirements

Efficiency

CSULB FF has many features that increase the efficiency of memory allocation and deallocation and user-experience performance. Below is a comprehensive list of all features that provide these benefits, with detailed descriptions:

Table View Controller cell reuse

iOS provides a convenient way to save memory when using table views by implementing a method called *dequeueReusableCellWithIdentifier*. This allows a cell with the specified ID to be "reused" when the cell isn't within the viewable portion of the screen. So instead of a large amount of cells being created and stored in memory, only a fixed amount need to be, saving time and space.

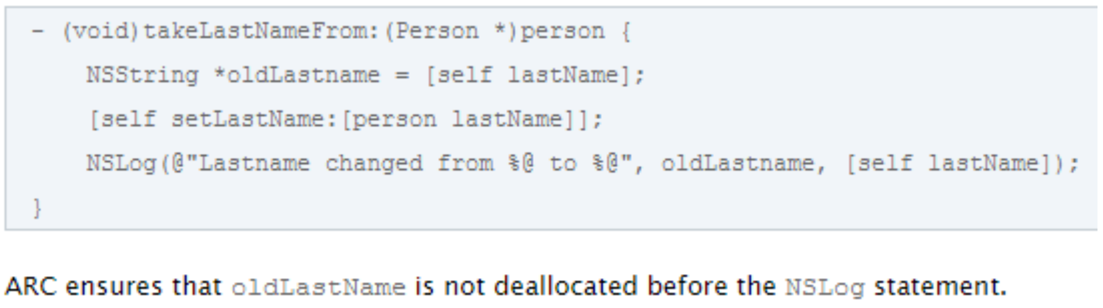
Parse query save/fetch in background

All Parse queries implement methods that fetch and save the requested objects in the background. These methods are non-blocking and allow the user to continue to interact with the GUI interface. This allows for improved application responsiveness.

iOS use of ARC

Apple’s iOS implements automatic reference counting (ARC), a novel way of managing memory and reducing battery load on mobile platforms, which often suffer from restricted memory sizes and battery life. ARC provides some of the benefits of garbage-collecting such as retain and release operations, without the inherent performance hit. It works by adding code at compile time to ensure that objects live as long as necessary, but no longer. To accomplish this, new lifetime qualifiers were added that help memory management.

- **__strong is the default. An object remains "alive" as long as there is a strong pointer to it.**
- **__weak specifies a reference that does not keep the referenced object alive. A weak reference is set to nil when there are no strong references to the object.**



```
- (void)takeLastNameFrom:(Person *)person {
    NSString *oldLastname = [self lastName];
    [self setLastName:[person lastName]];
    NSLog(@"Lastname changed from %@ to %@", oldLastname, [self lastName]);
}
```

ARC ensures that `oldLastName` is not deallocated before the `NSLog` statement.

Below is an example of ARC as it would behave in code::

Reliability

Description of Reliability Measures

CSULB FF's navigation is performed by the default GPS system found in all iPhones. This system provides many methods that are invoked when a user's location changes and has an accuracy of approximately 100 meters. These built in methods are guaranteed by the OS to function and include *didUpdateLocations*, and *startUpdatingLocation*.

Error/Failure Detection and Recovery

The only data that is stored locally is the key associated with the currently logged in user; therefore, most error detection and recovery is handled automatically by the Parse database. Local data "corruption" can be overcome by refreshing the application, logout/login procedures,or reinstalling the app. Noncritical errors are handled internally, with NSLog statements.

User alerts are displayed for the following severe errors:

- Failure to add a friend
- Failure to poke a friend
- Failure to authenticate user
- Failure to load augmented reality

Allowable/Acceptable Error/Failure Rate

CSULB FF has a minimal amount of allowable errors, mostly stemming from clashes between the app's need for permissions and a user denying those permissions. These include:

- No internet connection

    - This results in the user not having access to the application, since an internet connection is required for a user to authenticate.
- Denying Location Services
    - This renders the user invisible to any future location updates, and also prevents navigation to a friend as a current location is needed to calculate distance.
- Denying Push Notifications
    - This prevents the user from receiving any pokes from friends.
- Denying Camera access
    - This prevents the user from using the navigation feature because the augmented reality requires a live video feed in order to function

Security

Data Security

CSULB FF keeps data secure in two major ways: through password protected user accounts, and by secure communication through the various API's. **Authenticated Users** give security to local devices by allowing only registered users to access device services such as the camera and gps, as well as the CSULB FF app. **Secure Communication** occurs through the Parse and Facebook API. Queries to the Parse database can only be executed by authenticated users using a registered Parse application. The Facebook API facilitates CSULB FF's Facebook integration and handles the transition to display the secure Facebook login page.

System Design Overview

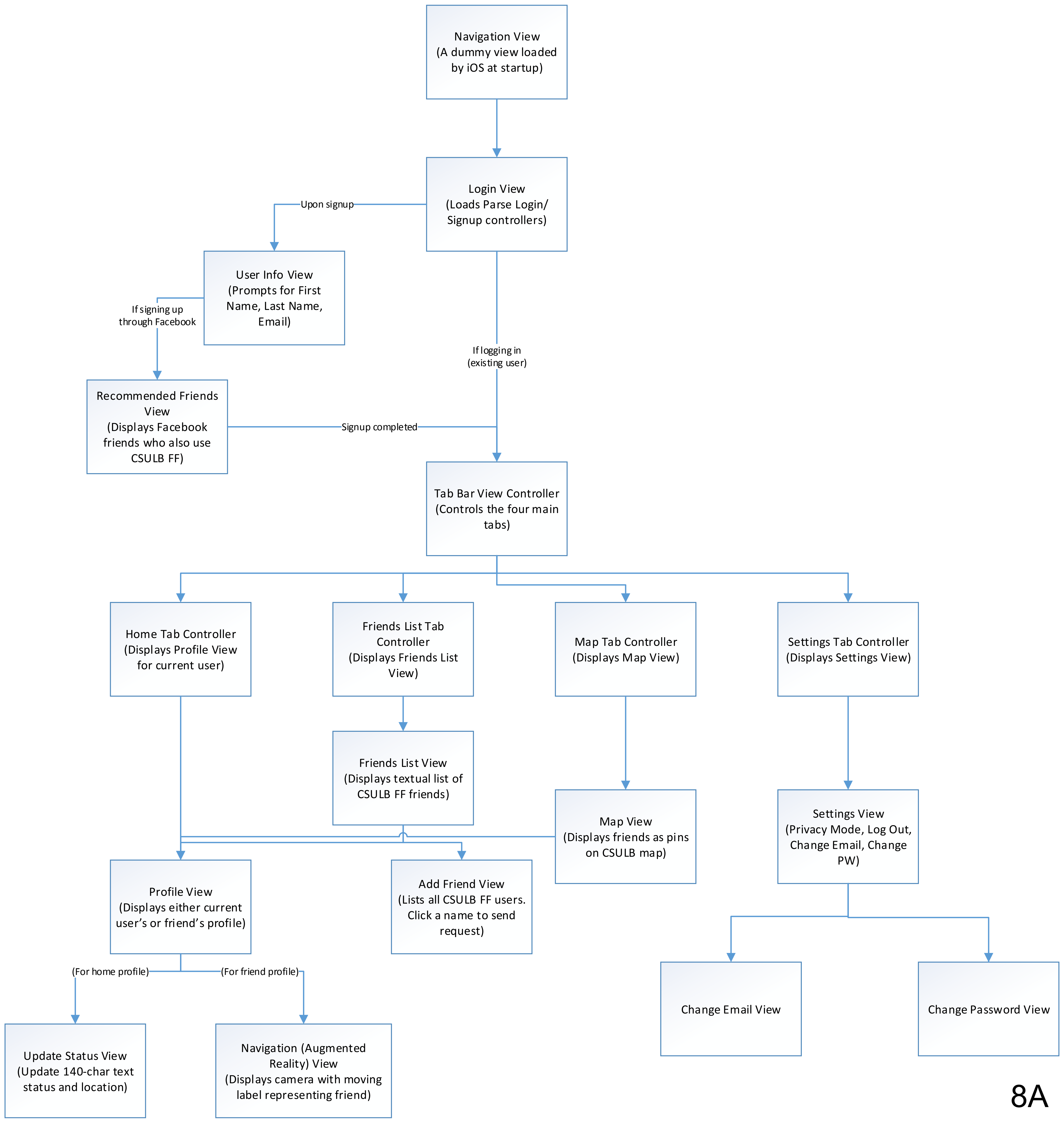
iOS Project Class Diagram

A table of the iOS project view controllers, and two utility classes, follows.

Login/ Registration	Main Tabs	Secondary Tabs	Parse GeoTools
Login (Loads Parse Login/Signup controllers)	Tab Bar (Controller for the four following tabs)	Update Status (Update 140-character status and location)	GeoPoint Annotation (Utility to create an iOS MapViewAnnotation using a PFGeoPoint)
User Info (Prompts for First Name, Last Name, Email)	Profile (Displays either current user or friend's profile)	Navigation (Displays camera with moving moving representing friend)	Location Translation (Utilities for closest building & checking if coordinates on campus or not)

Recommended Friends (Displays Facebook friends who also use CSULB FF)	Map (Displays friends as pins on CSULB map)	Change password	
	Friends List Table (Displays textual list of CSULB FF friends)	Change email	
	Settings (Privacy Mode, Log Out, Change Email, Change Password)	Add friend (Lists all CSULB FF users. Click a name to send a request)	

A class diagram of the iOS project view controllers follows.



Navigation View (A dummy view loaded by iOS at startup)
Login View (Loads Parse Login/ Signup controllers)
Upon signup
User Info View (Prompts for First Name, Last Name, Email)
If signing up through Facebook
Recommended Friends View (Displays Facebook friends who also use CSULB FF)
If logging in (existing user)
Signup completed
Tab Bar View Controller (Controls the four main tabs)
Home Tab Controller (Displays Profile View for current user)
Friends List Tab Controller (Displays Friends List View)
Map Tab Controller (Displays Map View)
Settings Tab Controller (Displays Settings View)
Friends List View (Displays textual list of CSULB FF friends)
Map View (Displays friends as pins on CSULB map)
Settings View (Privacy Mode, Log Out, Change Email, Change PW)
Profile View (Displays either current user's or friend's profile)
Add Friend View (Lists all CSULB FF users. Click a name to send request)
(For home profile)
(For friend profile)
Update Status View (Update 140-char text status and location)
Navigation (Augmented Reality) View (Displays camera with moving label representing friend)
Change Email View
Change Password View

8A

Description of System Operation

Actions are performed on the application by clicking buttons and entering text. Sometimes actions query from or update the backend database.

Implementation Languages

Objective-C was chosen over the recently released Swift language because more learning materials were available on the former. This was a good decision because potential jobs for Team 2 members may involve Objective-C.

System Data Structure Specifications

Other User Input Specifications

Identification of Input Data

The methods to input data are the iPhone keyboard and touch screen. Data from the former is contained in the Objective-C class NSString. Data from the latter is handled by iOS navigation controllers.

Source of Input Data

The user inputs come from the device's keyboard and screen. Specifically in the app itself, they will come from Text Fields and Buttons. The Text Fields are provided and the user types inside to fill out information specified. Buttons are available to click on once input is finished.

Input Medium and/or Device

CSULB Friend Finder was developed and tested on an Apple iPhone 5 running the iOS 7 and 8 operating systems. Other iOS devices that run iOS 7 or 8 are expected to work, but the appearance of the app may not be optimized for their screen dimensions.

Data Format/Syntax

Textual data such as names and usernames are stored in NSString. Locations (of CSULB buildings and of CSULB FF users) are stored in a Parse convenience class called GeoPoint, which encapsulates latitude and longitude fields. Users themselves belong to a Parse convenience class called User. The Parse-provided Login and Signup controllers integrate with User, managing authentication and Facebook integration.

Other User Output Specifications

Identification of Output Data

All outputs are displayed to the user via the mobile device's screen. When one clicks on the "Find your friend" button on a friend's profile view, one is presented with an augmented reality display consisting of: real-time video capture from the back camera (facing away from the user currently using the app), and labels depicting the friend to navigate to. The labels move around on the display as the user rotates his/her body, thereby providing pedestrian navigation capability.

System Database/File Structure Specification

CSULB FF stores data on the databases and servers of a framework called Parse. The Parse database is a key-value store with support for tables. Each table represents a Parse class (but is simply a PFObject inside Objective-C code). The tables used in CSULB FF are: User, Building, FriendRequest, and Friendship, which have self-explanatory names. Explanation follows in the section titled “System Data Structure Specifications”. Following is a description of each table.

User		
Column Name	**Type**	**Description**
objectId	String	Identifier unique to all Parse objects
username	String	Username
password (hidden)	String	Password (Hidden from viewing, but changeable)
authData	Facebook session token	16-digit Facebook account identifier
email	String	Email
first_name	String	First name
last_name	String	Last name
location	GeoPoint	GeoPoint with lat. and long.
status	String	140-character max
isOnPrivacyMode	Boolean	“True” if user’s location is to be hidden from map and on friends list, “False” if not
facebookId	String	Unique identifier used by Facebook API. Used to identify Facebook friends of a new user as recommended CSULB FF friends.

Building		
Column Name	**Type**	**Description**
objectId	String	Identifier unique to all Parse objects
name	String	Building name (e.g., “Liberal Arts 5”)

location	GeoPoint	Building latitude and longitude

FriendRequest		
Column Name	**Type**	**Description**
objectId	String	Identifier unique to all Parse objects
RequesterId	String	objectId of the User who sent the request
RequesteeId	String	objectId of the User to whom the request was sent

Friendship		
Column Name	**Type**	**Description**
objectId	String	Identifier unique to all Parse objects
Friend1_Id	String	objectId of one user in the friendship
Friend2_Id	String	objectId of the other user
Notes: Each friendship is represented by two rows, with the Friend1_Id and Friend2_Id switched. This design decision is due to the following situation: The ddFriend menu must list all CSULB FF users who are not already friends with the current user. Retrieving those users requires a Parse database query that filters out Users whose objectId is 'Friend1_Id' in a 'Friendship' with 'Friend2_Id' = the current user's ID, and also those whose objectId is 'Friend2_Id' in a Friendship with Friend1_Id. The Parse database is not a relational database, and does not support multiple boolean conditions on a field such as objectId. The alternative of representing each friendship by two rows was implemented so that the multiple boolean test on objectId is not needed, since the boolean condition will hold true for one of the redundant friendship rows.		

Modules Accessing Structures

The main iOS application uses all tables in the Parse database. For example, the Home tab uses the User table to display one's own Name, Status, and location on a map. The Friends and Maps tabs use the Friendship table. The Add Friend menu, accessible from the Friends menu, uses the FriendRequest table.

Module Design Specifications

Module Functional Specification

The main iOS Application's purpose is to handle the majority of the app, including navigation for the user and the overall user interface. The main purpose of the GeoLocations module is to handle the locations of each user and building, as well as calculating the distance between each object. The main purpose of the Augmented Reality module is to handle the AR display and interface and how it works with location coordinates. The main purpose of the login/signup and the Facebook login is to handle creating a new account just for the application and logging in with that created account or Facebook account.

Functions Performed

- Main iOS Application
    - The functions of the main iOS application include handling navigation and the main user interface and ensuring data transfer between views and other modules
- GeoLocations
    - The functions of GeoLocations module include handling longitude and latitude points for locations of both static buildings and application users, using an algorithm to determine the distance between two of these locations, and integrating these functions with the provided iOS maps
- Augmented Reality
    - The functions of the Augmented Reality module include handling the AR user interface, all of the data needed to run it, and the transition from the profile view to the user interface
- Login/Signup
    - The functions of the login/signup module include handling creating new accounts, logging in with those accounts, and handling data associated with each of those accounts
- Facebook login/signup
    - The functions of the Facebook login/signup module include handling all Facebook interaction (including but not limited to getting user data from Facebook servers), associating an application account with that Facebook account, and logging in with that created account

Module Interface Specifications

- Main iOS Application
    - The main iOS application module interfaces with all other modules. As a result, this module handles a lot of input/output arguments, which includes but is not limited to user locations and user data
- GeoLocations
    - The GeoLocations module requires
- Augmented Reality
    - The Augmented Reality module mainly interfaces with the GeoLocations module, as it requires multiple location coordinates and distance algorithm data in order for it to be displayed in the AR interface. It also requires input/output from the main iOS Application module, mainly for access to the user profile
- Login/Signup
    - The login/signup module interfaces with the main iOS application. It is responsible for creating a new user, and thus all of the data associated with a user (name, current location, etc.)

- Facebook login/signup
  - The Facebook login/signup module interfaces with the main iOS application and login/signup module. It needs to have access to Facebook data, so in order for this module to be used there must be a valid Facebook account associated with it. After that is done, an account is created using that information

Module Limitations and Restrictions

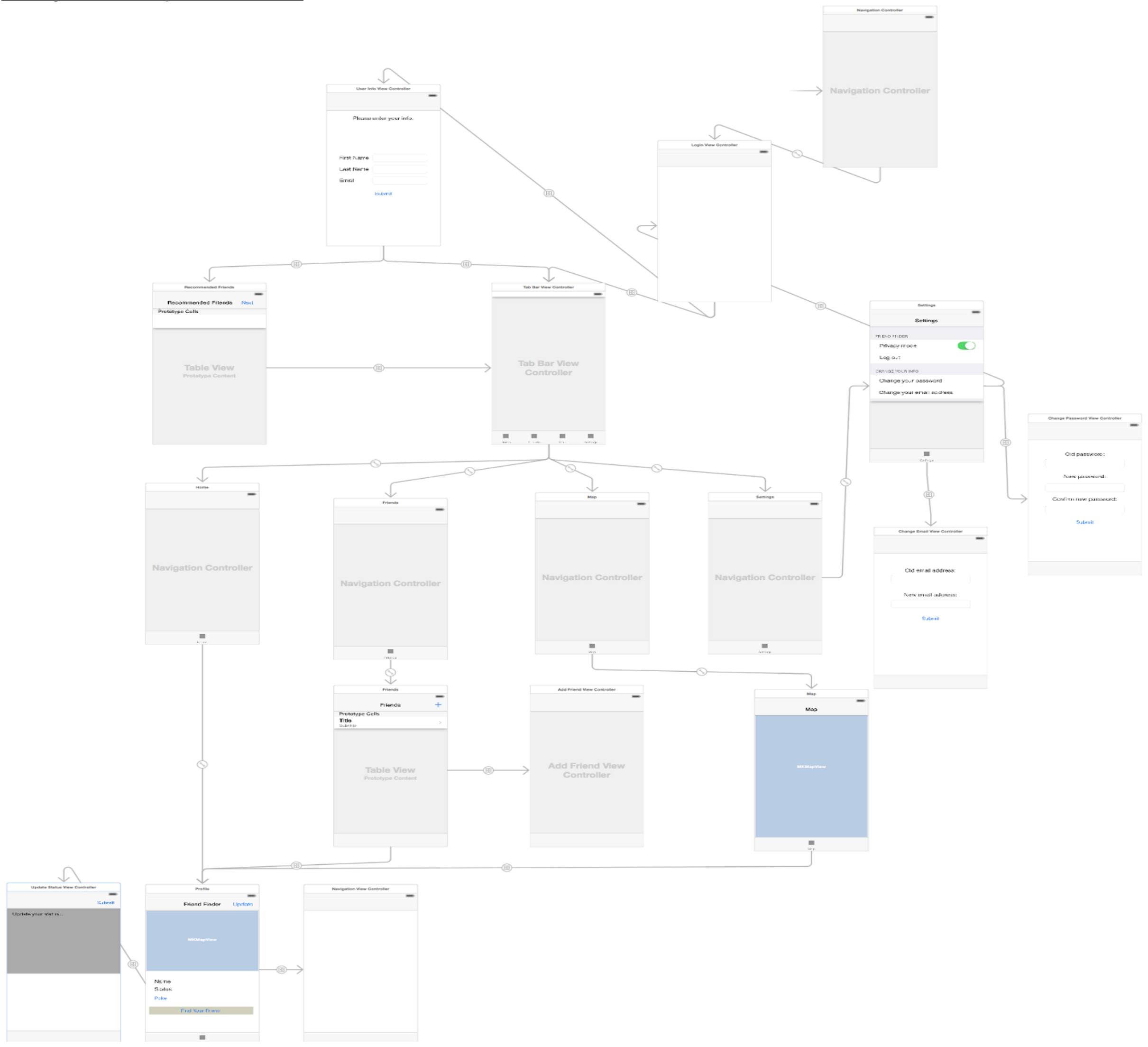
- Main iOS Application
  - Although the Main iOS Application handles the majority of the application, it does not complete specific and specialized functions by itself. It must rely on other modules for these functions
- GeoLocations
  - The GeoLocations module must have the longitude/latitude of the buildings hard-coded in, as there is no other way to get these coordinates. This means that for future plans, if we wanted to expand the area of our friend finder, we would have to hard code these new locations
- Augmented Reality
  - The Augmented Reality module has no major limitations or restrictions, other than the requirement for the user to have a working camera in order for it to be used.
- Login/Signup
  - The Login/Signup module has no major limitations or restrictions, other than relying heavily on our backend process
- Facebook login/signup
  - The Facebook login/signup module has no major limitations or restrictions, other than relying on Facebook servers and our login/signup module.

Module Operational Specification

Locally Declared Data Specifications

- Main iOS Application
  - The locally declared data for the main iOS application are user location and user data.
- GeoLocations
  - The locally declared data for the geo locations module are the longitude and latitude points.
- Augmented Reality
  - The locally declared data for the augmented reality are locations from the geoLocation module and distance calculated for how far another user is from you.
- Login/Signup
  - The locally declared data for login/signup is user (name,password,email)
- Facebook login/signup
  - The locally declared data for facebook login/signup are verifying if a user has facebook credentials in their account.

Storyboard Specification



Description of Module Operation

- Main iOS Application
    - The main iOS application module structures the user interface navigation with all the other modules. This allows the user to have a free flowing through modules.
- GeoLocations
    - The Geo Locations module holds the longitude and latitude coordinates of each user that is used to generate pin location.
- Augmented Reality
    - The Augmented Reality module uses the distance it calculated from the geolocation to display how far the friend is from the user in the camera.
- Login/Signup
    - The login/signup module allows the user to login or signup into the application.
- Facebook login/signup
    - The Facebook login/signup module allows the user to login or signup with facebook into the application.

Locally Declared Data Specifications

Because iOS applications follow the MVC (Model-View-Controller) software model, there are no unnecessary global variables. The only global variance is a singleton location manager, which geolocation capability to all views in the app. There are several locally declared variables, which exist as properties, in each view controller. The following describes the most important ones (excluding text labels, buttons, etc) briefly. Additional explanation lies in the source code files for each view controller.

One important view controller is the AppDelegate. It is the Model of the MVC model. AppDelegate contains the locationManager property. That is, the singleton for the entire app is stored in AppDelegate.

Another important view controller is ProfileViewController. This view controller can be used to display either the current user (in the home tab) or a friend (in the friends list or map views). There are multiple label and button properties in the view controller, such as “nameLabel”, “statusLabel”, etc. Of note is the “pokeButton” and “findYourFriendButton” properties. If the user to be displayed is the current user, then he/she should not be able to poke to navigate to him/herself. Therefore, these buttons are disabled by accessing their properties.

Algorithm Specification

No heavy computation is performed in CSULB FF. A typical algorithm creates a query to the backend database with the appropriate conditions, then loads that data into relevant displays. For example, the following is the most intricate algorithm of CSULB FF:

```
   PFUser *user = [PFUser currentUser];
   PFQuery *friendshipQuery = [PFQuery queryWithClassName:@"Friendship"];
   [friendshipQuery whereKey:@"Friend1_Id" equalTo:user.objectId];

   PFQuery *finalQuery = [PFQuery queryWithClassName:@"_User"];
   [finalQuery whereKey:@"objectId" doesNotMatchKey:@"Friend2_Id"
inQuery:friendshipQuery];
   [finalQuery whereKey:@"objectId" notEqualTo:[PFUser currentUser].objectId];
```

This algorithm first initializes a query for the Friendship class, searching for rows whose Friend1_Id is equal to the current user’s ID. Then, another query is initialized for the User class, for Users whose objectId does not match Friend2_Id in the friendship query just performed. This query is implemented within a delegate that implements a certain interface. The backend framework calls this delegate, running the query and loading it into a table of all users who are not friends with the current user, so that the current user can send them friend requests.

System Verification

Items/Functions to be Tested

Testing was performed on the modules: Main iOS application, Facebook login/signup, normal login/signup, GeoLocations, and Augmented Reality. Functions divided between these modules were: map view, profile view, augmented reality, logging in, registering for an account, resetting password, logging out, poking, adding friends, changing email, changing password.

Justification of Test Cases

Test cases are justified as all possible actions within the app. Therefore, test cases were tested manually, from the point of view of the user.

Test Run Procedures and Results

Test run procedures are done by actually clicking on the buttons on the screen and making sure they do the correct operations they are assigned to do.

Discussion of Test Results

Some test cases failed such as logging in/signing up, registering, having the closest building algorithm work correctly, and adding friends. Therefore, testing improved the quality of CSULB FF.

Test cases

Logging in/Sign up without Facebook

Test Case Number	LoginSignupWithoutFacebook1
Test Item	Accepting/Declining push notifications
Pre-conditions	The user loads the FF Finder app for the first time.
Post-conditions	The user accepts or decline push notifications for app.
Input Specifications	User clicks accept or decline on push notification.
Output Specifications	The user view changes to login view.
Pass	User is redirected to the login page.
Fail	The user is not directed to login page. The user push notification accepts when user declines notification. The user push notification declines when user accepts notification.
Assumptions/constraints	Assumption that you are connected to the internet. Assume the user has basic application knowledge to load the app.
Dependencies	none

Test Case Number	LoginSignupWithoutFacebook2
Test Item	Sign Up
Pre-conditions	The user is not a register user.
Post-conditions	The user view will move to the signup page.
Input Specifications	The user clicks signup button on the login page.
Output Specifications	The user view will change to the signup page.
Pass	Upon clicking the sign up button, user is successfully redirected to the signup view.
Fail	If the user clicks signup button and the view is not directed to the signup view.
Assumptions/constraints	Assume the user can hit the signup button with some basic application knowledge. Assume there is an Internet connection.
Dependencies	LoginSignup without Facebook passes.

Test Case Number	LoginSignupWithoutFacebook3
Test Item	Registering new user
Pre-conditions	The user enters a valid username, password and email.
Post-conditions	The user view will be directed to the FF Finder home page.
Input Specifications	User clicked signup button.
Output Specifications	The user view will change to FF Finder home page.
Pass	The user is redirected to the home screen.
Fail	If the user is a correct user, then the user doesn’t get redirected to home page.
Assumptions/constraints	Assumption that you are connected to the internet. Assume the user has basic application knowledge to hit signup button.
Dependencies	LoginSignup without Facebook2 passes.

Test Case Number	LoginSignupWithoutFacebook4
Test Item	Logging in by pressing the Login button for traditional login

Pre-conditions	The user is a registered user. The username and password are filled in.
Post-conditions	The view will move to the FF home page.
Input Specifications	The user inputs a valid username in the username box and valid password in the password box.
Output Specifications	The view redirects you to the home screen.
Pass	The user is redirected to the home screen.
Fail	The user is not a register user and not directed to FF home screen.
Assumptions/constraints	Assume the user can hit the signup button with some basic application knowledge. Assume there is an Internet connection.
Dependencies	Login/signup without Facebook3 passes.

Test Case Number	LoginSignupWithoutFacebook5
Test Item	Directed to the Friend Finder home screen.
Pre-conditions	The user enters a valid username and password. User logins successfully.
Post-conditions	The user view will be directed to FF Finder home view.
Input Specifications	Clicking signup button on the login page.
Output Specifications	The view will change to the signup page.
Pass	The user is redirected to the home screen, or you are told that you entered the wrong username/password.
Fail	The user is told that you entered the wrong username/password.
Assumptions/constraints	Assumes that you are connected to the internet. Assume the user has basic application knowledge to hit login button.
Dependencies	LoginSignup without Facebook(3 or 4) passes.

Logging in/Sign up with Facebook

Test Case Number	LoginSignupWithFacebook1
Test Item	Facebook Signing In
Pre-conditions	Clicking on initial Facebook Login Button
Post-conditions	The view will be directed to a Facebook Log In webpage.
Input Specifications	Clicking on the Facebook Sign In button.
Output Specifications	The view will change a Facebook Log In webpage.
Pass	Upon clicking the in app Facebook login button, user is successfully redirected to the Facebook Log In webpage.
Fail	____________
Assumptions/constraints	Assume there is an Internet connection.
Dependencies	None

Test Case Number	LoginSignupWithFacebook2
Test Item	Permissions View
Pre-conditions	User enters valid username and password.
Post-conditions	After logging in, the application will ask user for permission to specific information.
Input Specifications	Clicking Log in button on the Facebook site.
Output Specifications	The view will change to a permissions page.
Pass	Upon logging in successfully, user is presented with the permission view screen.
Fail	____________
Assumptions/constraints	Assume the user can hit the submit button with some basic application knowledge. Assume there is an Internet connection.
Dependencies	LoginSignupWithFacebook1 passes.

Test Case Number	LoginSignupWithFacebook3
Test Item	Personal Information View
Pre-conditions	User enters valid username and password. User submits successfully submits the requested

	information. User accepts CSULB FF permissions.
Post-conditions	The view will be directed to a Personal Information screen that will ask for first name, last name, and email.
Input Specifications	Clicking accept on the CSULB FF permissions.
Output Specifications	The view will change to a Personal Information View.
Pass	Upon clicking the button, user is successfully redirected to the Personal Information View.
Fail	____________
Assumptions/constraints	Assume the user can hit the submit button with some basic application knowledge. Assume there is an Internet connection.
Dependencies	LoginSignupWithFacebook2 passes.

Test Case Number	LoginSignupWithFacebook4
Test Item	Recommended Friends View
Pre-conditions	User enters valid first name, last name, and email. User submits successfully submits the requested information. User submits successfully submits the requested information. User accepts CSULB FF permissions.
Post-conditions	The view will move to the Recommended Friends screen.
Input Specifications	Clicking submit on the Display Personal Information Screen.
Output Specifications	The view will change to a Recommended Friends View.
Pass	Pass – Upon clicking the submit button, user is successfully redirected to the Recommended Friends screen.
Fail	____________
Assumptions/constraints	Assume the user can hit the submit button with some basic application knowledge. Assume there is an Internet connection.
Dependencies	LoginSignupWithFacebook3 passes.

Test Case Number	LoginSignupWithFacebook5
Test Item	Directed to the Friend Finder home screen
Pre-conditions	User hits next button on the Recommended Friends screen.
Post-conditions	The view will move to the home screen: Friend Finder.
Input Specifications	Clicking submit Next button.
Output Specifications	The view will change to a Friend Finder View.
Pass	Pass – Upon clicking the Next button, user is successfully redirected to the Friend Finder screen.
Fail	____________
Assumptions/constraints	Assume the user can hit the submit button with some basic application knowledge. Assume there is an Internet connection.
Dependencies	LoginSignupWithFacebook4 passes.

GeoLocations

Test Case Number	GeoLocations1
Test Item	Initialize a GeoPointAnnotation from a valid PFUser
Pre-conditions	The PFUser has valid fields: first_name, last_name, location
Post-conditions	The GeoPointAnnotation is initialized with: user's location as Coordinates, user's first+last names as Title, approximate location as Subtitle
Input Specifications	Call [GeoPointAnnotation initWithObject: user]
Output Specifications	
Pass	Checking with debugger, the GeoPointAnnotation has the required fields.
Fail	The GeoPointAnnotation is missing one or more fields.
Assumptions/constraints	The PFUser has well-formed fields: the first and last names are valid strings, and the location is a valid PFGeoPoint.
Dependencies	None

Test Case Number	GeoLocations 2
Test Item	Add a GeoPointAnnotation to a MapView
Pre-conditions	The GeoPointAnnotation has valid “coordinate”, “title”, and “subtitle” fields.
Post-conditions	The MapView displays the GeoPointAnnotation with the correct location, title, and subtitle.
Input Specifications	Call [mapView addAnnotation:geoPointAnnotation]
Output Specifications	
Pass	Click on the pin. The pin is in the correct location, and has the correct name and subtitle.
Fail	The pin is not shown, or: The pin is not in the correct location, or: Click on the pin. The pin does not have the correct title or subtitle.
Assumptions/constraints	The GeoPointAnnotation is valid. The mapView was functioning correct prior to placing the pin.
Dependencies	GeoLocations1

Test Case Number	GeoLocations 3
Test Item	Use LocationTranslation: closestBuilding to determine the closest CSULB building to a PFGeoPoint.
Pre-conditions	The PFGeoPoint has a valid set of coordinates.
Post-conditions	The closestBuilding method returns a string “Around …”, e.g., “Around Hall of Science”.
Input Specifications	Call [LocationTranslation closestBuilding:thePFGeoPoint]
Output Sepcifications	
Pass	The correct “Building name” string is returned. The tester can verify by looking at the coordinates on Google Maps.
Fail	A nil string is returned, or: An incorrect “Building name” string is returned.
Assumptions/constraints	None
Dependencies	None

Augmented Reality:

Test Case Number	AugmentedReality1
Test Item	Segue to navigation view
Pre-conditions	Clicking on the “Find Your Friend” Button. The destination view controller is initialized and passed a valid detailItem object.
Post-conditions	The segue is performed to the destination view controller.
Input Specifications	[dest setDetailItem:friend]
Output Specifications	The view will change to the augmented reality view, so that the user can start navigating
Pass	Upon clicking the “Find Your Friend” button, the user is segued to the augmented reality view.
Fail	If detailItem is invalid, an error message displays directing the user to try navigation later.
Assumptions/constraints	Assume there is an Internet connection. Assume the detailItem object is valid.
Dependencies	None

Test Case Number	AugmentedReality2
Test Item	Initialize PRARManager
Pre-conditions	Location manager is initialized and starting to update location.
Post-conditions	Camera layer is successfully initialized.
Input Specifications	call [[PRARManager alloc] initWithSize:self.view.frame.size delegate:self showRadar:true]
Output Specifications	A valid camera layer is returned.
Pass	The camera layer appears on the current view.
Fail	The screen remains blank (white) if the camera session fails to add video input from the AVCaptureDevice.
Assumptions/constraints	Assume there is an Internet connection. Assume the device has a working camera and the user has given permission to use it.

Dependencies	AugmentedReality1 passes.

Test Case Number	AugmentedReality3
Test Item	Start displaying the augmented reality graphical overlays.

Pre-conditions	PRARManager has been properly initialized.
	friendPRAREIement = @[@{
	   @"id" : @(0),
	   @"lat" : @(geoPoint.latitude),
	   @"lon" : @(geoPoint.longitude),
	   @"title" : friendName }];

Post-conditions	Call method startARWithData when didUpdateLocations is invoked.

Input Specifications	call [self.prarManager
	   startARWithData:friendPRAREIement
	   forLocation:currentCoordinates]

Output Specifications	Returns an AROverlay on the camera layer.
Pass	The camera layer appears on the current view.
Fail	Outputs an alert if startARWithData is passed invalid data, and then dismisses the view.
Assumptions/constraints	Assume there is an Internet connection. Assume the user has given permission to use location services.
Dependencies	AugmentedReality2 passes.

Main iOS Application

Test Case Number	MainiOSApplication1
Test Item	Switching tabs
Pre-conditions	The user loads the FF Finder app for the first time.
Post-conditions	The user accesses a different tab, and all related data is passed along successfully.
Input Specifications	The user clicks on a different tab other than the current one
Output Specifications	The user view will change to the desired tab selected, and related data will be passed along.

Pass	User is redirected to the selected page, with relevant and related data passed along.
Fail	The user is not directed to the selected page. Data is not passed along, or the wrong data is passed along.
Assumptions/constraints	Assumption that you are connected to the internet. Assume the user has basic application knowledge to load the app. Assume the user knows what each of the tabs are.
Dependencies	none

Test Case Number	mainiOSApplication2
Test Item	Select friend
Pre-conditions	The user has friends.
Post-conditions	The user view will move to the selected friend’s profile page.
Input Specifications	The user clicks on the friend from the friends’ list
Output Specifications	The user view will change to the selected friend’s profile page
Pass	Upon clicking the on the selected friend, the user view will switch to the selected friend’s profile page.
Fail	The user is not directed to the selected friend’s profile page.
Assumptions/constraints	Assume the user has at least one friend. Assume the user has access to the friends tab.
Dependencies	Switching tabs passes.

Test Case Number	mainiOSApplication3
Test Item	Adding friends
Pre-conditions	The user knows someone to add.
Post-conditions	The user will have added a new friend.
Input Specifications	The user clicks on the “+” button on the friends tab.
Output Specifications	The user will have added a friend
Pass	Upon clicking on the “+” button, a friend request

	notification will be sent to the selected the friend. After the selected friend accepts the friend request, that friend will appear on the friends list.
Fail	The user is unable to select a friend to add. The selected friend declines the friend request.
Assumptions/constraints	Assume the user has someone to add as a friend.
Dependencies	Switching tabs passes.

Conclusions

Summary

We developed CSULB FF to help our fellow classmates to communicate with each other and help students make it easier to meet up on campus. CSULB students will no longer be lost meeting up with their friends while using this application.

Problems Encountered and Solved

Our initial encountered problems for the CSULB FF application was the login page and implementing facebook integration at the beginning of development. To solve these problems we had to research the parse documentation and learn how to implement a fully functional login page with facebook. The next encountered problem for the CSULB FF application was the augmented reality and poking. We could not test this features properly until we got a developer account to test it on the actual iphone. The next encountered problems were small little bugs throughout code while during development. We solved these problems through regression testing.

Suggestions for Better Approaches to Problem/Project

One suggestion for better approaches to Problem/Project is plan to more frequent meetings. To work on and discuss the project. Another suggestion is to develop in a multi platform ide, rather than Xcodes which is limited Mac OSX.

Suggestions for Future Extensions to Project

On suggestion for future extensions to the project is implementing users schedules to show if a user is busy in class or free to meet up. Another suggestion is to port the application to Android OS. The final suggestion is to invest more time to further clean the code and work more efficiently. As well as adding documentation to help maintain the code in the future.

User Manual

The user manual is located on the CSULB FF website.