Iota Pi Application for iOS

Senior Project Final Report - Dr. Franz Kurfess

Deborah Newberry - 15 March 2017

Table of Contents

Project Summary

For my senior project, I opted to create a meeting organization app for my school's chapter of Kappa Kappa Psi, Iota Pi. Kappa Kappa Psi is a national honorary fraternity for college band members. We meet every Sunday night, and during these meetings we plan events (both internal and external) that aim to meet our goal of making sure that the Cal Poly band programs have their social, financial, material, and educational needs satisfied.

Technologies

- Developed Using:
    - **Computer**: MacBook Pro (late 2011) running macOS Sierra Version 10.12.2
    - **Software**: XCode Version 8.2.1
    - **Testing Service**: Apple iTunesConnect's TestFlight
- Final Application:
    - **Tested With**: iPhone SE simmulator and iPhone SE running iOS 10.2.1, as well as various user's iPhones (specs unclear)
    - **Language**: Swift 3.0
    - **Database and authentication manager**: Firebase
    - **Dependency manager**: Cocoa-Pods

Screens

In order to ease our process with parliamentary and organizational procedures, the app consists of six main screens:

Login

In order to automatically be able to only login members of Iota Pi, the login feature uses the <firstname>.<lastname>@iotapi.com email address that is automatically created by the chapter Webmaster.

Forgot Password

Before logging in, users can reset their password if forgotten. An automatically generated email will then be sent to their iotapi.com account (which is automatically forwarded to their @calpoly.edu email) and will include a link to reset it.

Account Creation

Users can create an account by filling out a form. Six fields are required: first and last name (in order to generate @iotapi.com email), their Nationals roster number (found via the official Kappa Kappa Psi site), their administrator privilege (by default "None", but can

be changed by another admin later), their education class, and their current membership status (for example, Active or Conditional, among many other official statuses Nationals deems valid). These are necessary for the app to function correctly for a user. Optional fields for roster purposes include birthday, local address, phone number, and many more. Their original password is given via a randomly generated 6-character screen in an alert, and then they are responsible for changing their password once they have access to their account.

Admin:

Once created, the President or Webmaster must validate their account on the “More” options screen. Until then, the user will receive a “needs validation” error when attempting to log in, and they will not appear on the roster or any other portion of the app.

Announcements

This section serves as a general information view for all members, and is the first visible screen upon starting the application (if logged in). It lists announcements made by administrators in order from most to least recent. Each announcement made expires after seven days, and is then transferred to the archived announcement section. After one year, the announcement is automatically removed from the database.

General:

Brothers can read and search through announcements by committee tags (Band Social, Brotherhood, Fundraising, etc) and an optional text phrase. They can search through the list of archived announcements ordered by date by a text phrase (but no committee filters).

Admin:

All administrators can create new announcements by writing a short title, a description, and optional committee tags. These are set to expire automatically in seven days, and when submitted they send out notifications to all members’ phones. Additionally, all administrators have to option of swiping left on active announcements to either permanently delete or force archive them, and swiping left on archived announcements to permanently delete them.

Calendar

This section includes a web view of the Google Calendar already set up by the chapter. There are no plans to make an add event feature as it seems out of scope for the project, but brothers will be able to see all events and their details.

Voting

The voting section is the most important part of the application. An important note to make is that by National’s rules, only **Active** and **Associate** members are able to vote on any topics. Therefore, any other membership status will not be able to see either type of vote. There are two different votes that are simplified through the addition of the app, both of which are automatically deleted from the database after a full year has passed:

HIRLy Nominations

These are used to reward brothers that have gone beyond the call of duty and represented a trait that is picked by the Brotherhood committee.
General:
Brothers are presented the trait and its definition. They select a brother's name from an alphabetical list of all brothers that have not already won HIRLy in the past school year and write a quick reason as to why they believe that brother deserves to win. The nominations themselves are kept anonymous, but each user will be marked as having submitted a vote as to prevent multiple submissions.
All brothers can also opt to look at a list of archived HIRLy votes. These list the date and trait of the vote, as well as a list of winners. Clicking on these winners leads the brother to view all of the reasons submitted for why they deserved to win.
Admin:
The Brotherhood committee chair and President are able to create new HIRLy votes and archive or permanently delete current open ones. If a new vote is created while one is already open, it automatically archives the current open vote. They can also swipe left to permanently delete archived HIRLy votes.

Anonymous/Current Votes

These are votes that are decided anonymously during the meeting. They only survive for about an hour before the results are automatically archived.
General:
Brothers can vote "Yes", "No", or "Abstain" on all anonymous votes. They must enter the correct randomly generated code to be able to vote on the topic. The brother is then marked as having submitted a vote, and the "Yes", "No", or "Abstain" counter is incremented based on their selection.
Admin:
The President and Parliamentarian have the ability to create votes and forcibly archive them. They open a vote by creating a summary and description (similar to HIRLy), and are then shown a randomly generated string of six characters that the brothers need to submit their vote. They are also the only ones capable of viewing the list of archived current votes (and swiping left to permanently delete them if desired).

Attendance

This section allows brothers two actions: checking into meetings and viewing the roster.

Check-In

Brothers use this screen to make sure they are marked as "present" during a current session. An important note to make is that by National's rules, only **Active** and

Associate members are able to check into a meeting. Therefore, any other membership status will not be able to see the check-in screen.
General:
Brothers enter the session code for the current meeting to check-in.
Admin:
The President, Recording Secretary, or Vice President can start the meeting. When created, a randomly generated session code is shown. This code allows everyone to check-in. After the meeting, an administrator can then end the meeting. They can then look at all of the archived meetings ordered by date to see the start and end times of the meeting, as well as the list and total number of brothers present.

Roster

Brothers can search through the database for a particular person by first and last name or nickname. This information is currently stored on a spreadsheet on the chapter's Google Drive and is available to all members.
General:
Tapping a brother's name gives the current user their full name, nickname, roster number, status, phone number, education class, SLO address, birthday, instrument, and expected graduating quarter. If they click on their own info, they have the ability to change all but their roster number and status.
Admin:
The President and Recording Secretary have the ability to change any of the above listed fields for any brother except for roster number. The Webmaster and President will also have the ability to change that brother's admin privileges and delete the user if desired.

More

This section includes items that don't fit into any of the above categories. Any user has the ability to go directly to their info (which is the same as their roster screen details). They can log out of the app, and they can follow a link to the iotapi.com webpage. They also are able to change their password here, which is especially important for those who don't want to remember the randomly generated one they received on account creation.
Admin:
The President and Webmaster will have the additional option of validating any users pending validation. They can validate one or more users at once, or swipe left to delete any pending user if they do not wish to validate them.

Summary of Administrators and Their Abilities

Screen	Pres.	Recording Secretary	VP	Webmaster	Parliamentarian	Committee Chairs
Login	✓			✓		
Announcements	✓	✓	✓	✓	✓	✓ - All
Voting - HIRLy	✓					✓ - Brotherhood
Voting - Current	✓	✓ *	✓ *		✓	
Attendance - Check-In	✓	✓	✓			
Attendance - Roster	✓	✓				

* - added after progress report to validate “chain of command” in the event the officer above them is absent and can’t host the meeting (President → Recording Secretary → Vice President)

Horizontal Prototype

The Horizontal Prototype was the UI wireframe created to show the desired flow and look of the application. Every screen the user can see was mocked up in as much detail as possible.
Link to General Horizontal Prototype
Link to Administrator Horizontal Prototype

Problems and Solutions

There were two minor problems I had while completing the Horizontal Prototype.

1. Inexperience with Graphic Design
   Problem: I've never taken a Graphic Design class, and I have limited to no experience with photoshop tools. I ended up using GIMP, which is a free photoshop-type application available for Mac. It took a long time for me to create screens at first, though as I went on it did get a little easier as I got used to the software.
   Solution: Thankfully, I had created a simpler Horizontal Prototype in my Android class using GIMP, so I had a little bit of experience with GIMP and a decent amount of experience with InVision. In addition, I took a UI class fall quarter as one of my tech electives, and I ended up using some of the concepts in that class in my wireframes. These may not have helped my experience using GIMP directly, but it helped to have a clearer vision of my goal as I experimented with the tools.
2. Limit of InVision Projects
   Problem: Unfortunately, the free version of InVision limits the amount of wireframes you can create. I hit that limit when I decided to create the administrator version of the prototype.
   Solution: I archived my old Android wireframe after I downloaded it. This opened up a prototype for me to use in InVision. However, I had to add yet another prototype when I created a wireframe for my UI Final Project, but at that point I had already finished the prototypes. Unless I try to edit the projects, I'm not threatened with losing them. In the meantime, I've saved offline versions of them to be safe.

Vertical Prototype

The Vertical Prototype is focused on implementing basic working versions of APIs needed to construct the final application. In short, this included linking the app with Firebase using read and write access as well as implementing the login system (and persistent login to prevent the user from having to login every time they start the app).

Problems and Solutions

There are a couple of issues I ran into while trying to implement a basic version of the app:

1. Firebase Connectivity
   Problem: It took me some time to get used to how to connect to Firebase. Setting it up was simple enough, but I had to search relentlessly through the implementation documentation to be able to read and write. Another issue I had was how to ensure that multiple users doing things like voting wouldn't mess up the data on Firebase.
   Solution: I ended up using the idea of services. I have a service for all of the major screens, and that service handles all Firebase-related reading and writing. This simplified my views and helped me simplify my Model-View-Controller setup. I also learned about transactions in order to allow multiple user input. These transactions ensure that data is updated in real time so that it isn't corrupted in the case of multiple user submission.
2. Calendar System Choosing
   Problem: Swift does not have an easy way to connect to Apple calendar, and I also wanted to be able to connect to the already existing Google Calendar the chapter has been using for years. I wanted administrators to be able to fully interact with the calendar by adding events instead of just viewing them.
   Solution: After brainstorming and looking through a lot of third-party dependencies, I opted to cut out the ability of adding events and simplified the screen to just a webview of the calendar. I used Google Calendar's existing XML embedding option.
3. Scope
   Problem: I got incredibly carried away with the scope of the Vertical Prototype. It's mainly supposed to show that I can access and add things to the database, but I wanted to be able to make it as visually similar to the final prototype as possible.
   Solution: I detailed in my progress report what problems I had and my scoping issue. Even though I was unable to demo at the end of fall quarter, doing more work than necessary fall quarter set me up for success in releasing my beta app this quarter.

Final Product

Structure

For my app's code, I aimed to structure it in the best possible manner via methods I learned in my app development courses and my internship this past summer. Two main organizational structures I want to explain are the code and the database.

Model-View-Service-Controller (MVSC)

The MVC structure for an application is incredibly common, and the Service component is an important addition for database calls. The Model serves as the app's representation of the database content. The View is, as implied, the visual component the user sees on their screen. The Controller is what connects the Model and Service to the View. In it, all of the logic required

to know what is supposed to be seen when is held. Finally, the Service handles all of the database calls and sends the necessary information to the Models and Controllers.

App Example

Let's say I wanted to create a new announcement. In the AnnouncementView, I would touch on the creation button and fill out the appropriate fields. After submitting it, The AnnouncementViewController would send my inputs to the AnnouncementService. This service would then take my inputs and create a new Announcement model. That model would be pushed to Firebase, and upon success would call a method via something called the AnnoucementServiceDelegate. This delegate is a protocol that all controllers using the AnnouncementService must inherit. The delegate decides the names of the methods that must be created in order to be accessed inside a callback function. After the delegate method is called, the AnnouncementViewController's implementation of the method would tell the AnnouncementView to update its data. Finally, the updated data would show the new announcement that was created on the AnnouncementView.

Firebase Structure

Firebase is a hierarchical database, which made it easy to update quickly. I created four main parents to hold all of the data, all which are the same as the models in the app: Announcements, Brothers, Meetings, and Voting (which includes the children HIRLy and CurrentVote). Inside all of these is the ID of the child, and inside that live the children data of that ID.

App Example

The Announcements data looks like:

Significant Problems and Solutions

Unfortunately, as wonderful as Firebase is, I found some major faults while trying to create administrative users for my app.

1. User Creation
   Problem: Firebase's createUser method automatically logs in a user. This was problematic for two reasons: (1) if an administrator needed to create a user, they would automatically be logged in as that user and (2) if a new user needed to create their account, they would automatically be logged in and have access to all of the data (especially if they marked themselves as having admin privileges).
   Solution: To combat this issue, I opted to revote an admin's ability to create a user and force new users to create their own account. Once the app successfully creates an account and its database information, I log the user out automatically. Then an active administrator has to go in and "validate" said user, which "isValidated" as true in that user's database content. They are then able to login and access the app.
2. User Deletion
   Problem: Firebase only allows an account to be deleted if that user is currently logged in. This made it impossible for administrators to delete other accounts.
   Solution: I mimicked my solution for user creation by setting a flag, "isDeleted" to true if an administrator wants to delete the account. If true, that user, although still living in the database, no longer shows up in the application's roster or any other information. The next time the user tries to log in, I use the moment they are authenticated to delete their account and database information. They are then sent a message that they have been deleted from the system, and are unable to log in.

Beta App

At the beginning of March, I decided that the app's current functionality needed to be beta tested before continuing. I enlisted the help of the chapter, and five brothers with iPhones agreed to help test the app's functionality. We met in a fishbowl on a Saturday afternoon in the library, and I sat down and walked them through account creation before letting them run wild. I created a Google Form for them to fill out, but unfortunately forgot to tell them to bring laptops, so I only had two post-testing session responses. However, I did get a lot of real-time feedback. I took notes on this and used the responses as a basis on which to improve the app for the second beta build.

TestFlight

TestFlight is Apple's beta testing service. Unfortunately, student developer accounts don't have access to it. After struggling with third-party testing services and realizing that if the app were to go into the app store at some point that a student developer account can't do that either, I

decided to fork up the $100 fee to become an official app developer. However, my issues did not end there. For one, it took a lot of reading, testing, and failing to figure out how to get the proper certificates for my app, as well as how to submit builds to iTunesConnect, which is the official hub for preparing applications for the iTunes app store.

Beta Review

Another huge hurdle I had was the fact that Apple's TestFlight had two variations of beta testing - internal and external. Internal testing was incredibly easy to set up, but was only available for other developers. Because of this, I needed to use the external testing option, which came at the hefty price tag of waiting for a "beta review." This review is ambiguously strict (and hated) by most developers, as its guidelines and timeline vary. I managed to get my first build approved for testing relatively quickly (in an hour so). However, build 2.0.1 took several hours to be reviewed, and was then rejected on the basis of "bugs found" with no more information. I had already made improvements at that point, so I went ahead and submitted build 2.0.2 to be reviewed. After *fifteen hours*, the app was accepted, and I was then able to release it for the users that beta tested for me before to be able to try and break.

Conclusion

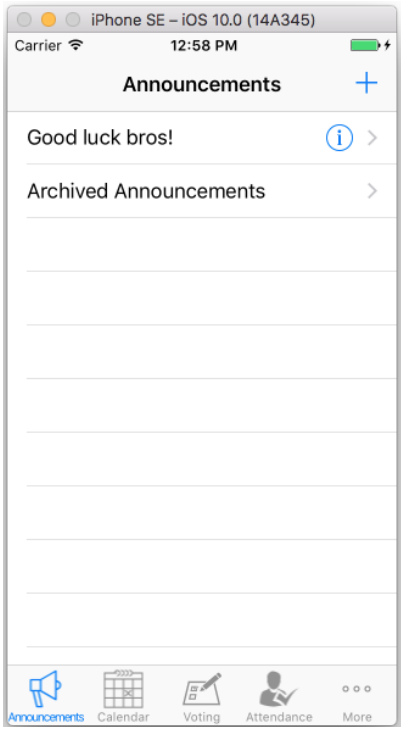
After a long six-month process of building the app from scratch, I'm happy to report that I've gotten incredibly positive reviews of it from brothers. They seem incredibly interested in how it was created, and agree that it would be very useful. However, without an Android counterpart, it can't be used by all brothers and will therefore not be integrated until then. I have already heard from several brothers that they would be interested in building the Android app, but until then, I have opted to give the application to the chapter itself. That way it can be built upon and improved as needed, and if there comes a time when it can be used, the chapter will own all of the content and App Store access. I'm doing this by attaching a separate iotapi.kkpsi@gmail.com account to Firebase and as a contributor to the Github Repository. This will be given to the chapter webmaster to be passed down between the years.
Link to Github Repository

Screenshots

Here are some screenshots of the current app versus their prototype counter parts, as well as some of the current screens I was unable to show before. I'm using an open-source Cocoapods dependency to create the alerts, and a different open-source dependency to create the extensive account form. There will be a readme in the Github repository that will give credit where due for all the tab bar icons I used as well.

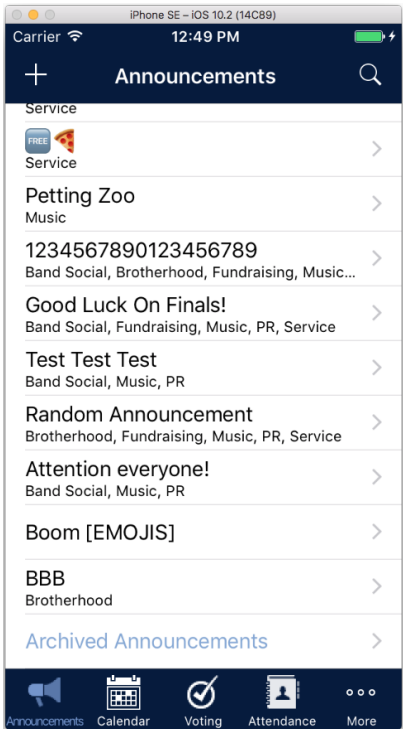
Prototype v. Current

Prototype

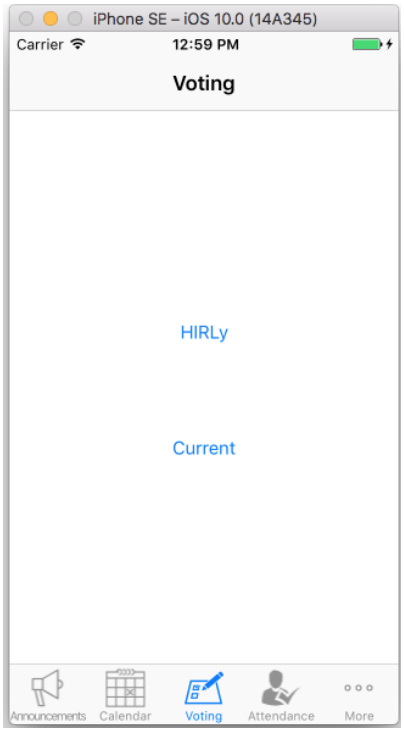


Announcements

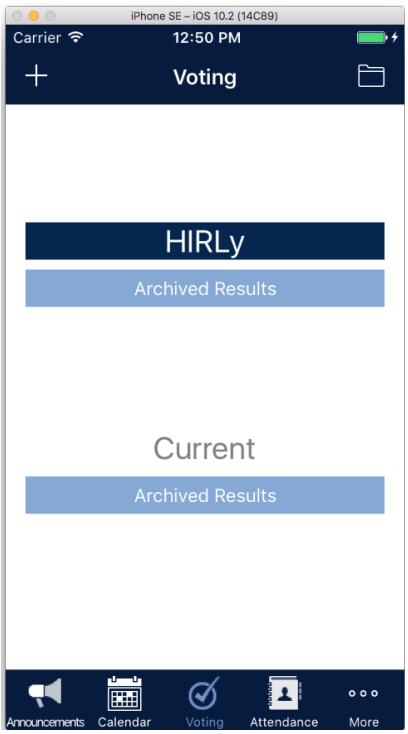
Current



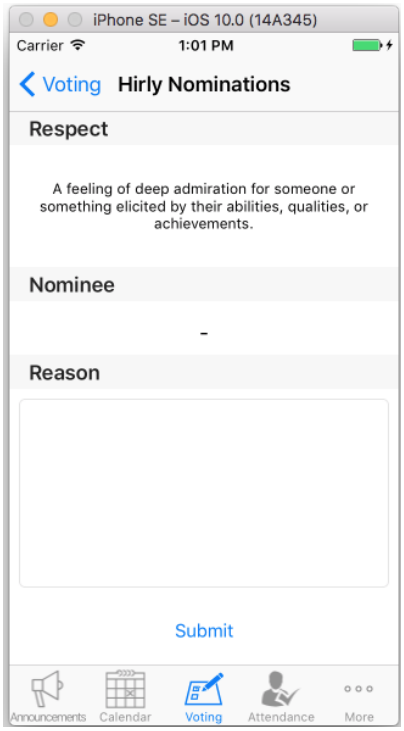
Announcements



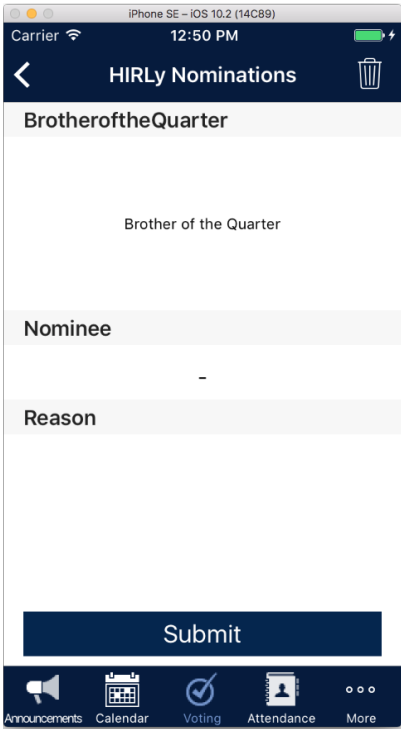
Voting - Selection



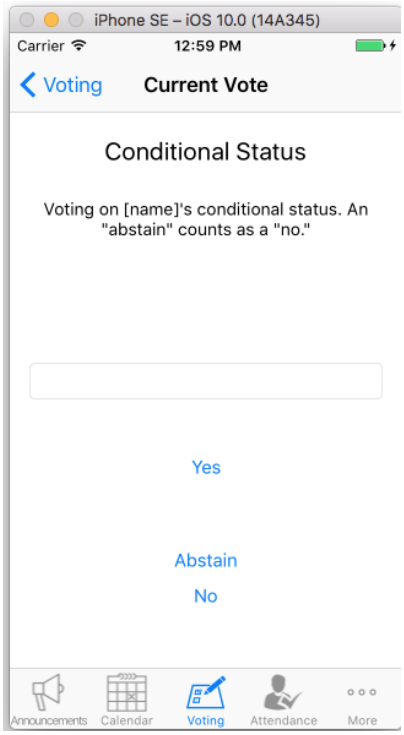
Voting - Selection



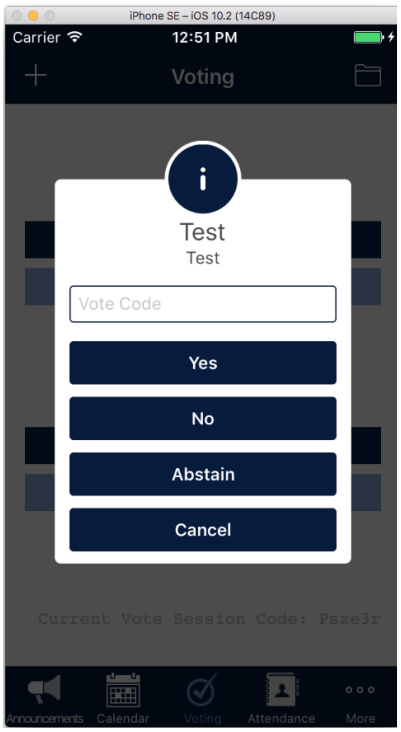
Voting - HIRLY Form



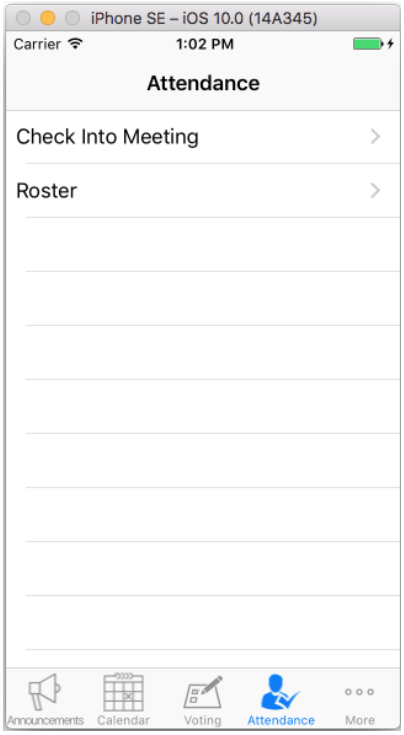
Voting - HIRLY Form



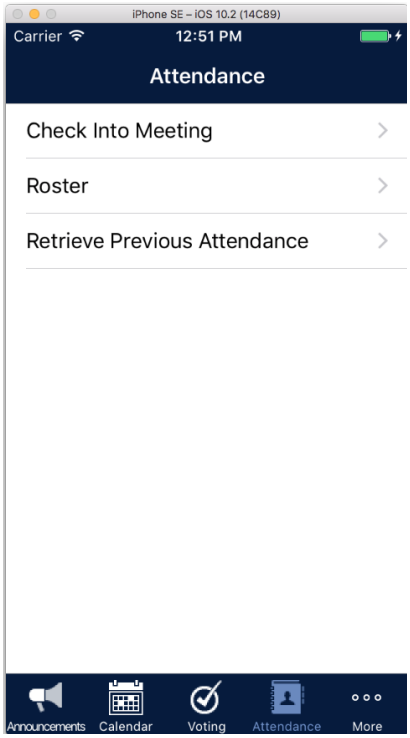
Voting - Current



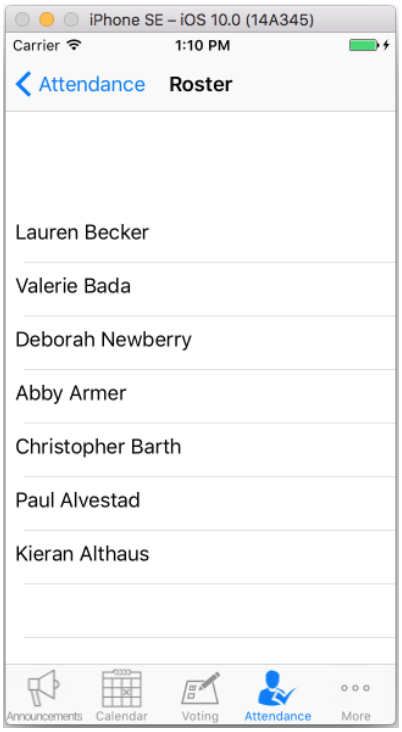
Voting - Current



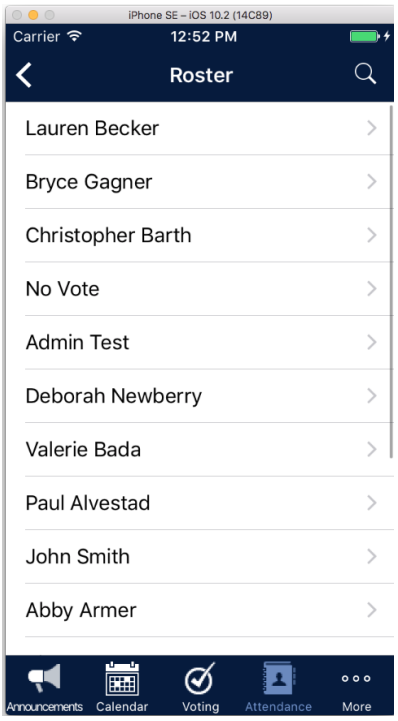
Attendance - Selection



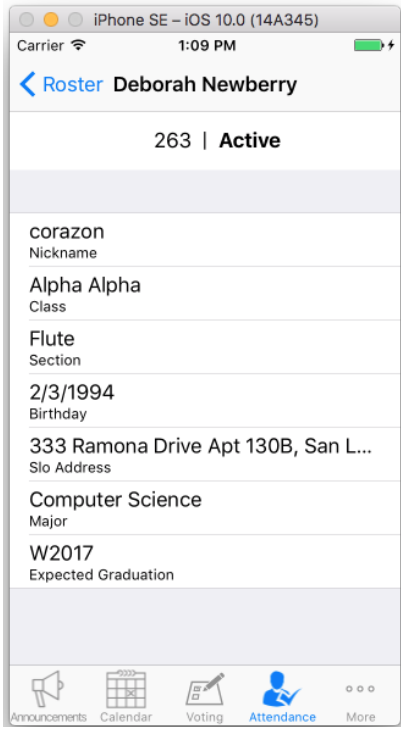
Attendance - Selection



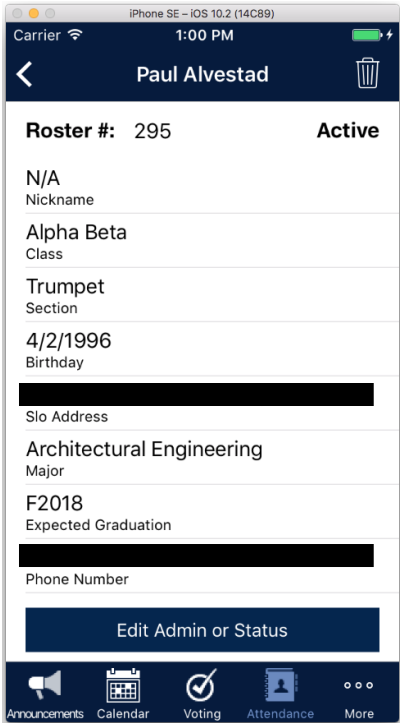
Attendance - Roster



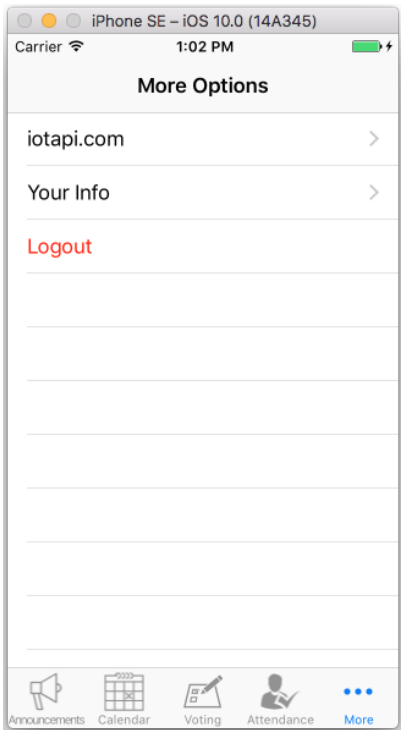
Attendance - Roster



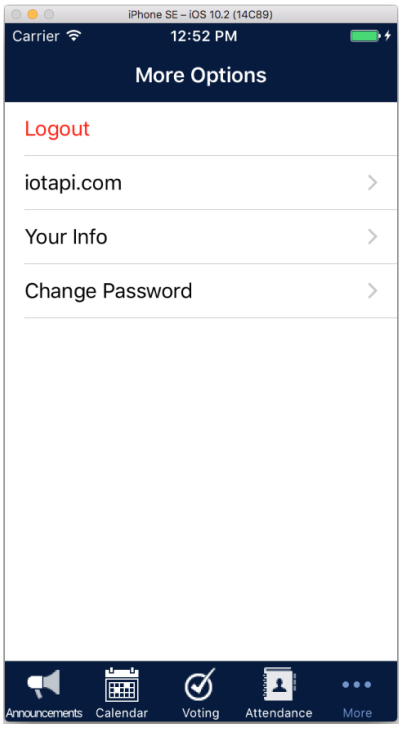
Attendance - Brother Info



Attendance - Brother Info

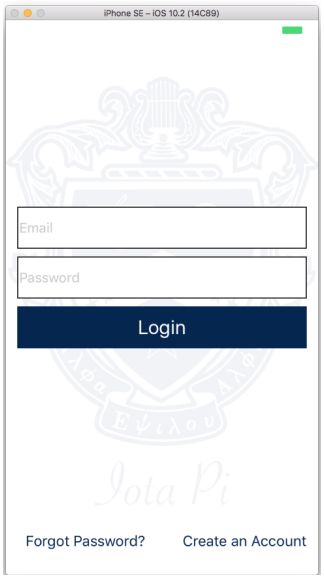


More

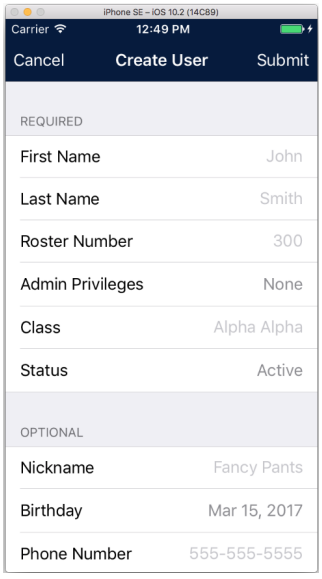


More

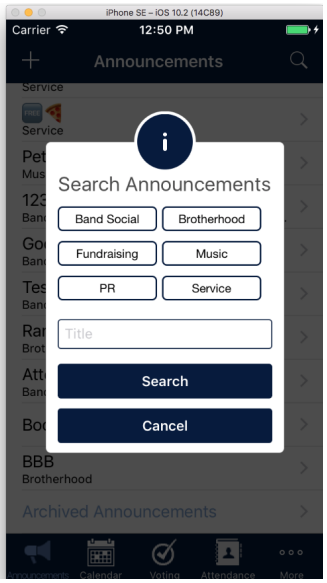
Other Screens



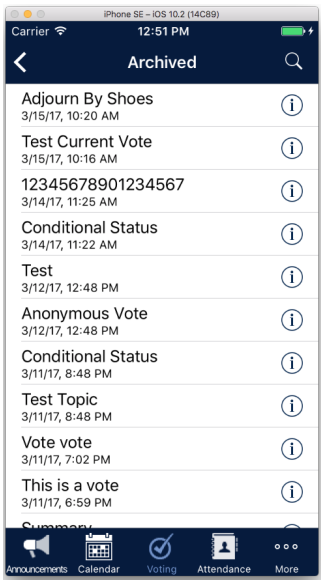
Login



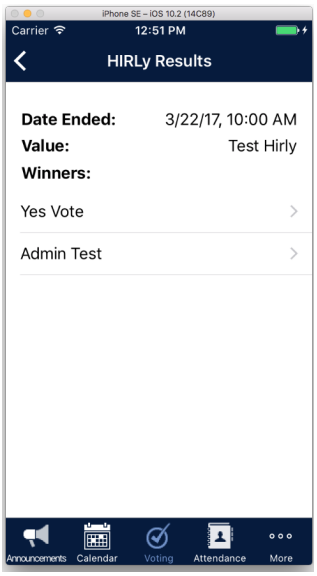
Create Account



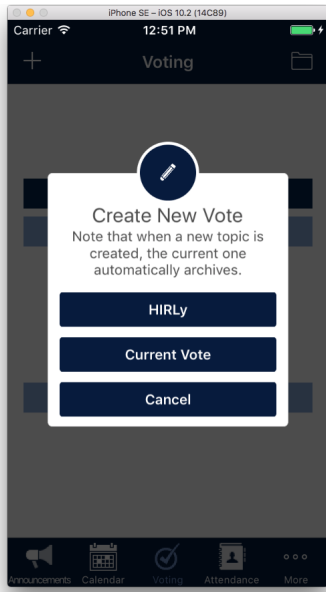
Announcements - Search



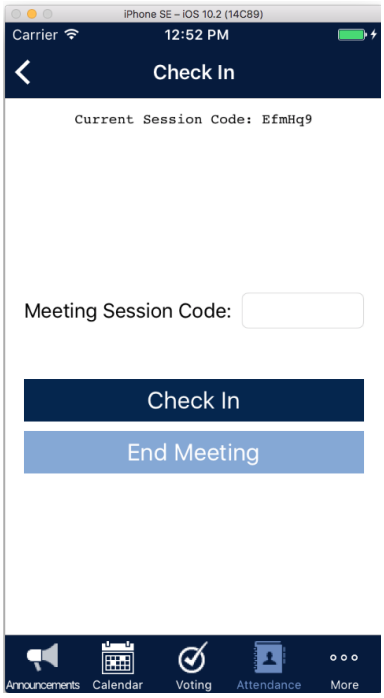
Voting - Archived Current Votes



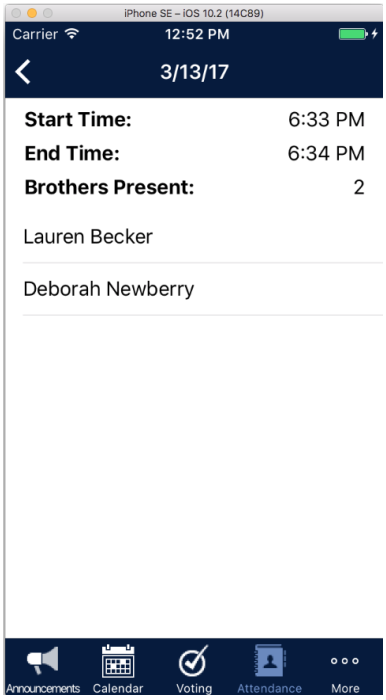
Voting - Archived HIRLy Info



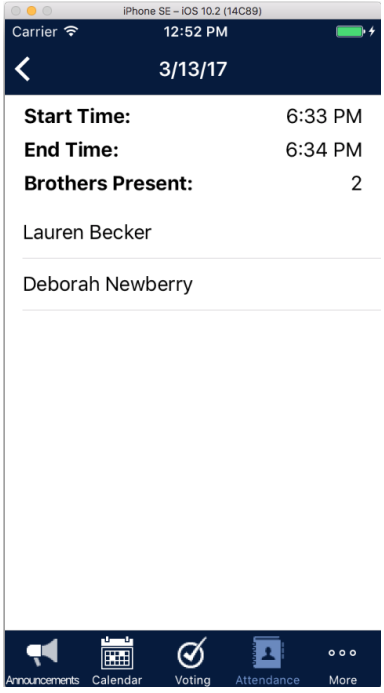
Voting - Create New



Attendance - Check In



Attendance - Archived Meeting Info



Attendance - Archived Meeting Info